MINISTRY OF NATIONAL EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Traffic Control System in the Connected Cars Context (TCS-3C)

LICENSE THESIS

Graduate: **Andrei TUDORICA**

Supervisor: **Prof. dr. eng. Rodica POTOLEA**

2019

MINISTRY OF NATIONAL EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

DEAN,
Prof. dr. eng. Liviu MICLEA

HEAD OF DEPARTMENT,
Prof. dr. eng. Rodica POTOLEA

Graduate: **Andrei TUDORICA**

Traffic Control System in the Connected Cars Context
(TCS-3C)

1. **Project proposal:** *The domain if Intelligent Transportation Systems (ITS) is accountable for improving the mobility context by creating safer, smarter and eco-friendlier solutions. The existing software solutions, on the other hand, fail to freely offer the possibility of centralized research and development. The scope of this project is to propose (design, implement and test) a software solution able to integrate several activities performed in the Connected Cars Context. Furthermore, the project also aims to study the effects of the decisions made by traffic participants on the traffic context, using a simulator module.*

2. **Project contents:** *Introduction, Objectives, Bibliographic Research, Analysis and Theoretical Research, Implementation, Testing and Validation, User's Manual, Conclusions, Bibliography, Appendix 1*

3. **Place of documentation**: Technical University of Cluj-Napoca, Computer Science Department

4. **Consultants**: Msc Student Dan TODERICI**,** Phd Student Ioan STAN

5. **Date of issue of the proposal:** November 1, 2018

6. **Date of delivery:** July 10, 2019

Graduate: ______________________________

Supervisor: ______________________________

MINISTRY OF NATIONAL EDUCATION

TECHNICAL UNIVERSITY
OF CLUJ-NAPOCA, ROMANIA

FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE DEPARTMENT

Declaraţie pe proprie răspundere privind
autenticitatea lucrării de licenţă

Subsemnatul(a)____
____,
legitimat(ă) cu ________________ seria ________ nr. _____________________________
CNP ____, autorul lucrării
____
____
____elaborată în vederea susţinerii examenului de finalizare a studiilor de licență la Facultatea de Automatică și Calculatoare, Specializarea ____ din cadrul Universităţii Tehnice din Cluj-Napoca, sesiunea _________________ a anului universitar ___________, declar pe proprie răspundere, că această lucrare este rezultatul propriei activităţi intelectuale, pe baza cercetărilor mele şi pe baza informaţiilor obţinute din surse care au fost citate, în textul lucrării, şi în bibliografie.

Declar, că această lucrare nu conţine porţiuni plagiate, iar sursele bibliografice au fost folosite cu respectarea legislaţiei române şi a convenţiilor internaţionale privind drepturile de autor.

Declar, de asemenea, că această lucrare nu a mai fost prezentată în faţa unei alte comisii de examen de licenţă.

In cazul constatării ulterioare a unor declaraţii false, voi suporta sancţiunile administrative, respectiv, *anularea examenului de licenţă*.

Data ______________________

Nume, Prenume _________________________________

Semnătura

Table of Contents

Chapter 1. Introduction

1.1. Context

Over the last few years the number of cars has increased significantly, as more and more people use their personal vehicles in order to commute to work. For example, 87.9% of America’s daily commuters use their private cars, states [1]. As the number of cars increases constantly, the currently developed road networks cannot keep up with the demand. This creates traffic congestions to an unprecedented level. The most affected areas are, of course, the urban ones. [2] presents the effects of the traffic congestions in the biggest cities in America. People in Washington D.C. have an average of 74 hours of annual delay per commuter, while an average commuter in America is stuck in traffic for 34 hours every year. The effects are visible in the financial field as well, as the average US commuter loses $1400 by idling away gas, states [3].

A common practice is increasing the road capacity of the road segments that are the most demanded. This solution works on short term, but, as [4] states, these road segments are rapidly saturated due to induced demand. Some governments discuss, even if they do not implement it yet, a set of financial solutions that include increased taxes for traffic participants that want to enter highly demanded areas, during peak hours, as envisioned in [1]. The most effective solutions so far were the technological ones, such as Google Maps or Waze, that help traffic participants avoid highly crowded areas in their commute. These solutions spread the commuters over the entire road network more evenly. The algorithmic solutions of these products are secret, difficult to study and clearly extremely hard to imitate.

The current traffic context needs to be managed in a different manner, by taking into consideration every aspect of the traffic context. As the automotive industry has achieved a number of technological advances, more and more cars become connected. With this in mind, software solutions are the best option for managing the Connected Cars Context, states [5].

1.2. Problem Statement

The context of connected cars has been a main point of interest for the traffic researchers. Studies and experiments have been conducted to improve different topics of the context, such as routing, data collection, data interpretation, etc. Some of these works even obtained good results. Unfortunately, these studies, simulations and experiments have been done separately and rarely put together in order to create a bigger picture. This happens because the researchers don’t have a common platform to facilitate their experiments. During this thesis presentation, a novel solution to the problem of managing the Connected Cars Context is presented as a conceptual architecture and an implementation.

Table 1-1 presents a better organized problem statement.

The problem of	Inexistence of complex management and research systems for the connected cars context
Affects	Traffic participants, Traffic researchers, Traffic supervisors
The impact of which is	Incomplete management of the connected cars context Traffic congestions and delays in urban area transits
A successful solution would be	Open to more relevant features and allow them to work together Versatile for research

Table 1-1 Problem Statement Table

1.3. Resulting Product Position Statement

The Traffic Control System in the Connected Cars Context (TCS-3C) is a modular system that aids the routing, management and research of the Connected Cars Context. It brings together the members of the context under the roof of a single software solution, in order to make the Connected Cars Context evolve faster and more naturally. A better organized form of the resulting product position statement is presented in Table 1-2.

For	Traffic participants, Traffic researchers, urban traffic managers
Who	Need a better connected cars context management in an easy to use traffic aiding system
The TCS-3C	Is a distributed system
That	able to help routing, supervising and researching the Connected Cars Context
Unlike	Google Maps and Waze
Our product	Is open for the development of modules that could integrate not only the current number of cars in the area, but also infrastructure, previous traffic data and traffic supervisors' limitations. It also allows the connection of separate interfaces for traffic participants, supervisors and researchers.

Table 1-2 Resulting Product Position Statement

1.4. Structure of the project

The following chapters from this documentation describes the following aspects:

- **Chapter 2** contains the problem objectives and the two types of requirements that this project has, functional and non-functional.
- **Chapter 3** describes specific research done concerning the subject of navigation systems. It covers different existing solutions in the domain, describing their advantages and disadvantages. After that, the subjects concerning maps and data is tackled, followed by traffic data and routing elements.
- **Chapter 4** motivates the choice of the specific open source system. Additionally, it presents the conceptual design of the proposed system, with a practical example. Afterwards, it describes the design of the implemented solution, emphasizing the way the traffic data is modeled and how the routing is done.
- **Chapter 5** gives further details about the implemented design. It presents each component from two perspectives: the design and the flow of the component. This chapter presents the technical details of the implementation.
- **Chapter 6** presents the testing and validation metrics of both the implemented system and the proposed simulation purpose. The simulation scenarios are presented in this chapter and the results obtained for the system and for the simulation.
- **Chapter 7** illustrates how a new user can setup and run each component of the system as a starting point for future development or just for testing purposes.
- **Chapter 8** covers the problem objectives and the way they were addressed. It also comprises ideas of future related work both on the implemented configuration of the system and on any project that might use the same conceptual architecture.

Chapter 2. Project Objectives

2.1. Project Objectives

In what follows, the main objectives of the project will be defined. The objectives are organized as follows:

- The main objectives marked with **Ox**, x being the number of the objective
- The secondary objectives, that build up the main ones, marked with **Ox.y**, where x is the number of the main objective and y is the number of the second objectives
- Each objective can have one or two challenge tags. The **(CD)** tag marks the domain specific challenges and the **(CS)** tag marks the scientific specific challenges. These tags are added at the end of every objective.

O1: The first objective of this project is designing and implementing a system able to have a live overview of the Connected Cars Context. This system can be used in taking different actions related to traffic by several actors.

O2: The second one is developing a simulator module in order to feed data to the system and test the functionalities. This simulator will mimic genuine traffic participants actions related to traffic.

O3: Design and implement test suites which will help in the analysis of the effects of traffic participants decisions in traffic.

O1.1: Study the state-of-the-art existing systems, that tackle Connected Cars Context related problems. **(CS)**

O1.2: Design a high-level architecture of the system in its basic form. This would be the schema to follow over all next steps. **(CS+CD)**

O1.3: Decide upon what elements should be implemented from scratch and what elements should be used from external sources. The balance of these two needs to ensure that the capabilities of the system are maximal and no useless effort is performed in the development phase. **(CD)**

O1.4: Decide on one or more open source solutions to cover the elements that were decided to be imported in the last step. **(CD)**

O1.5: Setup the open source project in order to ensure that it performs the needed actions in a satisfying manner. **(CD)**

O1.6: Understand all the features, capabilities and limitations the open source project has, in order to use it at its maximum potential. **(CS)**

O1.7: Decide what elements of the open source solution will be used and in which components of the system. **(CD)**

O1.8: Implement the system with respect to O1.2 and O1.7 **(CD)**

O2.1: Decide what elements of the traffic will be simulated in order to test the system developed at **O1. (CS)**

O2.2: Design the simulator module in order to cover all the traffic elements that were selected at **O1.2**. **(CS+CD)**

O2.3: Implement the previously designed simulator, connect it to the system and tests its functionalities. **(CD)**

O3.1: Decide on the behaviours that are to be studied. **(CS)**

O3.2: Decide on a set of parameters used in order to simulate the chosen behaviours for the study. **(CS)**

O3.3: Decide on a list of metrics to study the simulated context. **(CS)**

O3.4: Implement the research part of the simulator, based on the features, parameters and metrics defined in the previous steps. **(CD)**

O3.5: Compute and study the results of the simulations under different test parameters. **(CS)**

O3.6: Draw relevant conclusions concerning the study. **(CS)**

2.2. Requirements

2.2.1. Functional Requirements

Based on the presented objectives, the functional requirements resemble the features of the project. Each requirement is shortly described, but more details about the design and implementation of these features can be found in chapters 4 and 5.

- Component able to hold the status of the traffic and update it when necessary. This component would be essential for the system as it would be the main Connected Cars Context representation in the system.
- Component able to generate routes based on different factors. This module would be used in the majority of activities that could be performed using the system.
- Component able to simulate the traffic context. This component is important as it assures the testability of the project. It is also the main tool that would be used to perform research activities.
- Simulator should be able to display statistics over the length of the entire simulation. This would be the interface that gives an insight of the activities of the system, ensuring everything works as designed.
- Simulator should output detailed statistics at the end of the simulation. This would aid the research activities, as well as studies on what concerns the capabilities of the system.

2.2.2. Nonfunctional Requirements

- **Response time.** The time it takes for the system to answer requests. After a request is sent it shouldn’t take the system more than ten seconds to answer and in average the response time should be below one second.
- **Scalability.** The technologies chosen for the implementation and the techniques used for manipulating data allow the system to scale easily with the increase in computational power. Even if in the implemented simulator the requests are serialized, the system is able to process parallel requests too. In a stronger environment the system might be able to compute over a million requests per second.
- **Resilience.** Due to the intensive work done for fixing all the problems that might appear and which are well documented in deploying an operative version of the

open source, levels of resilience are ensured. Also, all functionalities proposed in this thesis are being tested and bugs are closely analyzed and fixed before the final demo.

- **Reusability.** The system is developed in such a way that any developer that would like to use or modify it wouldn't encounter any difficulties. The essential snippets of code are commented and well documented in what follows.
- **Legal and licensing issues.** On what concerns the legal point the system was built using only open source external. Issues of this kind would have appeared if proprietary systems would have been used in the development of this system.
- **Testability.** This is one of the most important nonfunctional requirements on the list. There is even a functional requirement that covers the field. The testability of the system is proven by the results obtained using the simulator module.
- **Documentation.** This document is the primary proof that this requirement has been fulfilled. It contains all the necessary information for a new user to understand the purpose, design, architecture, implementation and how to use this system. Besides this document, the code contains explanations of the important snippets of code in the form of comments.
- **Adaptability.** This project runs on any desktop device, with no changes to be made to the code. Although the backend part that is also comprise in a repository that is set up and used for testing and development in a desktop environment, but it can be easily transferred to the mobile version being based on Java code.
- **Cost.** The project scope is this diploma thesis and a research paper is a welcome result, the main cost is time. Starting late December and continuing up to late June, the project will take up to six months to design and develop. Additionally, due to not having a log time tool in which the activity could be tracked, the exact time converted in hours will have small adjustments and is not exact, being evaluated in weeks or months rather than days or hours.
- **Data Integrity.** Even if traffic data is simulated, the information regarding the maps are as consistent and as accurate as they can possibly be. This is ensured by the fact that they are created from OpenStreetMaps resources, which enforced this requirement on their data sets and this property is maintained by constant updates.
- **Deployment.** All the code is accessible and also instructions regarding deployment of this project are provided in the user manual chapter, where there are instructions for deploying every component of the implemented system. Furthermore, all the main parameters that have to be set are described.
- **Disaster recovery.** The system stores its status frequently so in case of anything it can be redeployed in the same context. Every error that should appear while running is logged and can be studied afterwards while debugging.
- **Environmental protection.** As each traffic participant sends only their behavior and position to the Live Traffic server, there is no personal information to be put at risk over the network. Each traffic participant is represented in the simulation by a unique ID.

- **Source code escrow.** All our source code and documentation are to be found in a private GitHub repository and it can be available for viewing by the members of the research department and also by the license thesis committee.
- **Effectiveness and Modifiability.** The system tries to come up with a more effective way of storing and managing the live traffic. The implemented system can be easily modified as it allows adding other modules in order to implement additional functionalities.

Chapter 3. Bibliographic Research

3.1. Navigation Systems

A navigation system is an electronic system that facilitates the navigation. They are composed of both software and hardware. Navigation systems can be offline, online or a combination of both.

Offline navigation systems assume that all the data is on board of the using party and doesn't need any kind of aid from remote components.

Online systems have components that are located in fixed points and the using party communicates with it in order to receive the navigation information needed.

The hybrid systems can have the map information stored locally but still request routes from a remote component with higher computational power.

3.1.1. Proprietary Navigation Systems (Closed Source)

A proprietary software is a kind of software that doesn't allow anybody but the developing team to see or modify the source code. This kind of software is usually developed by companies that sell that product or offer it for free but generate profit out of it.

The most used proprietary navigation systems are the ones that are offline. These come integrated with the car and are provided by the car manufacturer or can be purchased separately as GPS devices from separate manufacturers. These systems are able to provide routes, direction, traffic information (e.g. speed limits) even in the absence of an open internet connection.

On the other hand, with the development of the mobile devices, the online navigation systems got to increase their popularity and reach. By benefiting from internet connection, these systems are able to offer the user a wider range of maps. They can also provide way more information on what concerns the current traffic status in certain areas, reviews and even street-level photos. The most relevant examples of online navigation apps are Google Maps and Waze. The fact online factor allows these apps to use user location in order to map the traffic and it allows them to use user data in order to show traffic information like traffic jams, police locations and many others. They offer the possibility to build a route in different ways: shortest distance, shortest time, least consumption, etc. Even if they have all these features, they still take into consideration only the user's choice when it comes to routing. They do not try to improve the entire context, but only the requesting users' route, even if they have one of the largest Connected Context when it comes to maps and navigation.

Considering the above mentioned, we are not able to use these systems directly in order to study and improve the Connected Cars Context.

3.1.2. Open Source Navigation Systems

An open source software is a type of software that are licensed by their developers in such a way that it grants users access to the code and it allows them to study, change and distribute the software, no matter the purpose. Most of these software systems are developed in a collaborative manner, many individual developers helping the process of designing, writing and testing the product.

These kinds of projects are the best choice for somebody that wants to try new routing techniques or even trying a new approach in the context of navigation systems. Writing a navigation software from scratch is a huge project and usually not worth it as the number of open source projects is big enough that it is almost sure that one can find a good starting point for their approach. Most of these projects are designed in such a way that they can be built over.

For this project, from the research of the open source community projects, two of them stood out.

- *OsmAnd*

  OsmAnd, presented in [6], is an open source software that runs on Android and iOS devices and is able to process OpenStreetMap data fully offline. OsmAnd has multiple navigation features like turn-by-turn voice guidance, traffic warnings (stop signs, pedestrian crosswalks, speed limits, etc.) and it has features for pedestrians and bicycle riders as well.

  The software was developed by a team of over 500 members and the entire source code is available on GitHub [7] under a license that allows any developer to modify and distribute the project, as long as they do it under the same type of license.

  The international community that takes part in the development of this product offered a huge collection of functionalities and elements on maps from over 40 countries, but it also brought a large set of undocumented features. The different coding styles approached by the developers affect the clarity of the projects code and makes it difficult for someone to build over it.

- *Itinero*

  Itinero, presented in [8], is an open source project built for route planning that was originally designed for logistical optimization. Just as OsmAnd [7], it processes OpenStreetMap data fully offline and it offers step by step instructions in routing. It can provide routes for cars, bicycles, pedestrians, mixed or custom profiles. On top of that, it can be used both as a library and a routing server for other projects, making it perfect for the development of other traffic related projects. As Itinero is open source, its entire code is available for free on GitHub [9]. The fact that this project is built by a single person ensures that the project is uniformly written and consistently documented.

3.2. Maps and Data

In order to develop an open source navigation system, an essential element is the map. This map needs to be free to use and needs to come in an accessible format. In this case maps offered by Google Maps or other similar proprietary systems are not useful.

One of the best, if not the best, open source map providers is OpenStreetMap. The official website can be found in [10]. OpenStreetMap is a free and editable map of the world that is released with an open-content license. It is built from scratch by volunteers, making it a prominent example of volunteered geographic information.

The OpenStreetMap data comes in several formats. The main elements used in the structure of the maps are:

Name	Description
Nodes	Representation of a specific points on the earth’s surface, defined by their coordinates (latitude, longitude)
Ways	Ordered lists of multiple nodes that define polylines. These are representations of linear features such as roads.
Relations	Multi-purpose data structures that document relationships between two elements
Tags	Used to describe the meaning of the elements they are attached to

3.3. Routing

3.3.1. Graphs

Formally, a graph is a pair of sets (V,E), where V is the set of vertices and E is the set of edges, as stated in [11]. Usually, when defining a road network, the set of edges represent road segments and the set of vertices represent the intersection of two or more road segments.

Route computing in road networks is a subject extensively discussed and studied. The route planning is usually done in weighted, oriented graphs. In order to model the features of most importance to a certain routing, the shape of the graph might change. As discussed in [12],when it comes to computing routes for pedestrians or bikes there ar not too many restrictions in the graph, compared to computing routes for cars.

When computing routes for motorized vehicles in road networks multiple restrictions have to be taken into consideration. One of the most frequently encountered restriction is „banned turn”. [13], [14] and [15] tackle the subject of routing techniqus when the aforementioned restriction is taken into consideration. From the research aforementioned, two types of graphs are presented, for modeling 2 different representations of the road network.

- **Node based graphs** are the type of graphs in which each edge models a road segment and has an assigned weight representing, for example, the distance of that segment. This is the usually utilised representation of a road network.
- **Edge based graphs** are the type of graphs in which road segments are modeled as nodes and turns between two consecutive segments

as edges. This type of approach in modeling the road network allows the incorporation of turn restrictions.

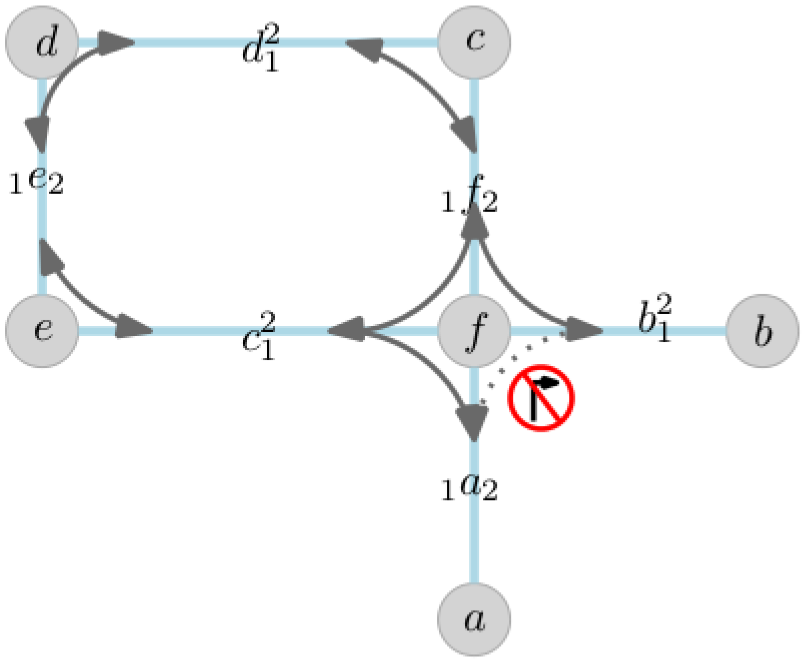


Figure 3-1 Visual Representation of an Edge Based Graph (from [16])

The edge based modelling technique has an advantage over the usually used nde based aproach when it comes to considering turn restrictions in the routing process. It can also reduce the number of nodes in the graph while increasing the number of edges, with the effect of speeding up the routing process, as stated in [17].

3.3.2. Algorithms

- **Dijkstra's algorithm** is an approach for finding the single-source shortest path between the nodes of a weighted graph. The technique was invented in 1956 by Edsger W. Dijkstra.

  Even if Dijkstra's algorithm is a greedy one, the optimality of the solution is ensured by the relaxation techniques used in its implementation. The output of the algorithm is a shortest path tree (SPT), rooted in the starting point of the search. This algorithm holds two sets of vertices, the ones already discovered and the ones yet to be discovered.

```
DIJKSTRA($V$, $E$, $w$, $s$)
INIT-SINGLE-SOURCE($V$, $s$)
$S \leftarrow \emptyset$
$Q \leftarrow V$            ▷ i.e., insert all vertices into $Q$
while $Q \neq \emptyset$
    do $u \leftarrow$ EXTRACT-MIN($Q$)
        $S \leftarrow S \cup \{u\}$
        for each vertex $v \in Adj[u]$
            do RELAX($u$, $v$, $w$)
```

Figure 3-2 Pseudocode of Dijkstra's best first search algorithm

The time complexity of the original Dijkstra's algorithm was $O(V^2)$ and it was based on a min-priority queue. There as certain variants of this algorithm that include the use of the Fibonacci Heap in the implementation of the min-priority queue which have the complexity $O(E + V\, log\, V)$.

- **Bidirectional Dijkstra's** approach is an improvement of the original algorithm that imply that the search is done in two directions. The first direction is from the start point forward to the final point (forward search). The second direction is from the destination point backwards to the starting point (backward search) [18]. This replaces the single search graph with two smaller graphs, one for each direction of search. The search stops when the two developing graphs touch each other.

  After stoppage, the route is constructed by adding to the elements of the branch from the backward search to the already constructed forward search tree.

```
$s_{pr} \leftarrow g(s)$
$t_{pr} \leftarrow g(t)$
$Open_F \leftarrow \{s\}$
$Open_B \leftarrow \{t\}$
**for all** $n \in V - \{s,t\}$ **do**
    $n_{pr} \leftarrow \infty$
**end for**
$p \leftarrow \infty$
**while** $Open_F \neq \emptyset$ AND $Open_B \neq \emptyset$ **do**
    $prmin_F = get\text{-}min(Open_F)$
    $prmin_B = get\text{-}min(Open_B)$
    **if** $prmin_F + prmin_B + \epsilon \geq p$ **then**
        return path for $p$
    **end if**
    **if** Forward frontier is expanded **then**
        $n = delete\text{-}min(Open_F)$
        $Closed_F = Closed_F \cup n$
        **for all** $succ \in n_{successors}$ **do**
            **if** $succ \in Closed_F$ **then**
                continue
            **else**
                $priority \leftarrow pr(succ)$
                **if** $succ \in Open_F$ **then**
                    **if** $succ_{pr} > priority$ **then**
                        $succ_{pr} = priority$
                    **end if**
                **else**
                    $succ_{pr} = priority$
                    $Open_F \leftarrow Open_F \cup \{succ\}$
                **end if**
            **end if**
            **if** $succ \in Open_B$ AND $g_F(succ) + g_B(succ) < p$ **then**
                $p \leftarrow g_F(succ) + g_B(succ)$
            **end if**
        **end for**
    **else**
        //Expand backward frontier analogously
    **end if**
**end while**
return failure
```

Figure 3-3 Pseudocode of Bidirectional Dijkstra

3.4. Communication Methods

Hypertext Transfer Protocol (HTTP) is an application protocol for distributed and collaborative systems. HTTP is the most widely used transfer protocol of the World Wide Web. In the client-server communication, HTTP is a request-response protocol.

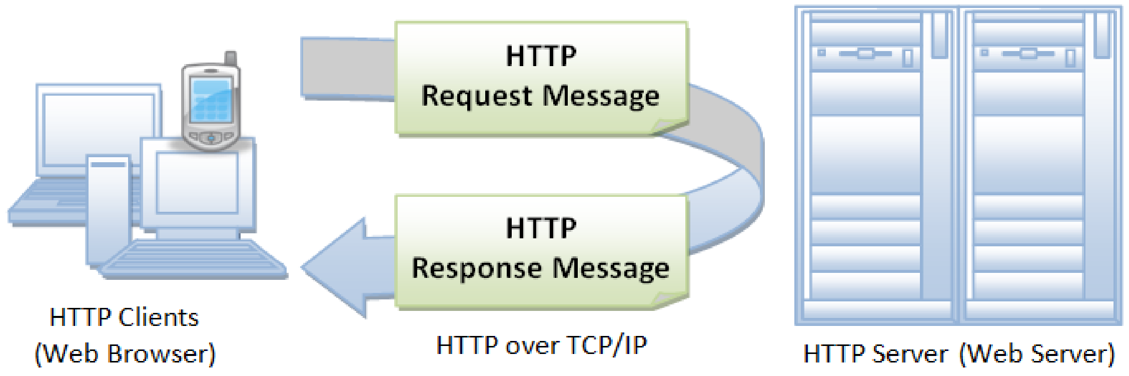


Figure 3-4 Visual Representation of the HTTP Protocol, [19]

- **HTTP request** is a message sent by the client in order to initiate an action on the server. A call contains in its start-line 3 elements: the HTTP method, the request target and the HTTP version.

The HTTP protocol defines a couple of methods to be used in communicating the type of action the server should be performing. The HTTP methods are presented in Table 3-1.

Name	Functionality
GET	Requests a representation of a specific resource
POST	Requests the server to accept the entity enclosed in the request
PUT	Requests the server to store the resource enclosed in the request
DELETE	Request the server to delete the specified resource
HEAD	Requests the server for a response identical with the one for a GET, but without the body
TRACE	Signal that is echoed by the server so the client can detect changes in the path a request takes to the server
OPTIONS	Requests the server to communicate what HTTP methods it accepts
CONNECT	Converts the request connection to a transparent TCP\IP tunnel
PATCH	Request the server to partially modify a resource

Table 3-1 HTTP Methods

- **HTTP response** is a type of message the Server sends, after receiving and interpreting a HTTP request.

  The HTTP response contains three elements: a status line, zero or more headers and, optionally, a message body. These elements contain all the needed information for the Client to receive the requested data or to confirm the action requested by the Client.

  The HTTP protocol defines five classes of responses, separated by the status line.

Class 100	Information Responses
Class 200	Successful Responses
Class 300	Redirection Messages
Class 400	Client Error Responses
Class 500	Server Error Responses

Table 3-2 Classes of HTTP Responses

Chapter 4. Analysis and Theoretical Foundation

4.1. Motivation for choosing Itinero

There are multiple reasons for which Itinero was chosen for developing this diploma project.

First of all, Itinero offers the possibility to be used as a library in any .NET project. That means the route computing is extremely customizable and can be used even for our very specific route computing criteria. It also means that the data structures used along the distributed components of this project are exactly the same, using as little effort to communicate as possible. The fact that Itinero is more of a tool than an application itself, allows the further development of a distributed system that uses it in all its components to be very easily extended.

Second of all, the way Itinero manipulates the map information is very well implemented, making it lightweight memory wise and fast even in the scenario of a distributed system, like in the case of this diploma project. Every data structure offers custom serialization and deserialization, which makes Itinero really fast when it comes to loading and transferring data.

Moreover, the fact that Itinero was developed by a single person (with confirmed design, algorithmic and development skills) ensures that the project is uniformly written. The code flows are consistent and well implemented and coding style is the same in the entire project, so the entire software is comfortable to study and understand, even if it is really heavy when it comes to the content of the code.

To complete the above stated argument, Itinero comes with a good documentation, both in the wiki page and in code comments. Because of this, understanding it and finding answers to almost any technical questions regarding Itinero is not as difficult as it might be in the case of other projects. Besides that, as this diploma project required some really specific tools from Itinero’s kit, whenever something was not clear enough, Ben, the developer of Itinero, was kind enough to provide the needed help.

Last but not least, the technical stack used in the development of Itinero, .NET framework written in C#, completed the list of arguments for taking this decision.

4.2. Conceptual Architecture of the System

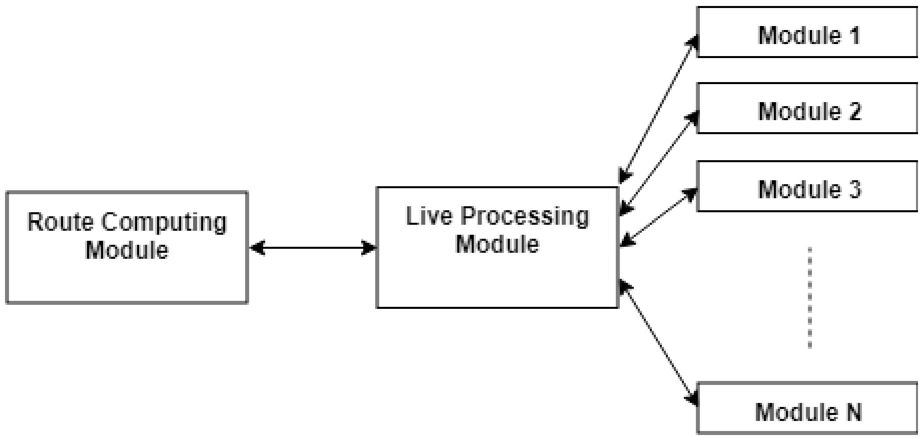


Figure 4-1 Conceptual Architecture of the TCS-3C

As a distributed system, the currently described project is built using modules. The main components are the Live Processing Module and the Route Computing Module. All other possible modules (either Action Based or Query Based) would communicate directly to the Live Processing Module. All these types of components will be described in the following subchapters.

4.2.1. Live Processing Module

The Live Processing Module is the core of this system. It is the module that supports or controls the actions of every other component in the system. It also holds the current state of the traffic.

This module has 2 types of input: route requests and notifications relating traffic. These inputs are received from Action Based Modules (4.3.2.). This module can also request traffic information from Query Based Modules, in order to keep the traffic status, it holds up to date.

When a route request is received, the Live Processing Module requests the latest traffic data from the available Query Based Modules (4.3.2.), in order to ensure the reliability of the currently held traffic data. Then it makes a request to the Route Computing Server for a route and after it receives the route, it sends it to the requesting module.

4.2.2. Route Computing Module

The Route Computing Module is the component of the system that takes care exclusively of computing routes, considering only the c2c infrastructure. It receives as input a route request and it returns the computed route based on the current status of the traffic. Whenever a route request is received, this module loads the traffic data that was updated by the Live Processing module.

4.2.3. Modules

- **Action Based Modules** are the type of modules that take direct actions in the traffic context. These modules are usually Client applications for Traffic Researchers, Traffic Participants or Traffic Managers. These modules are the ones that provide the input to the Live Processing Module and they process the responses received, described later on, in 4.2.4.

- **Query Based Modules** are the type of modules that hold a traffic status of their own, based on their own input. They receive requests from the Live Processing Module. They provided their current status to the Live Processing Module and it will take this information into consideration when updating the general traffic status.

4.2.4. Example of build system on the conceptual architecture

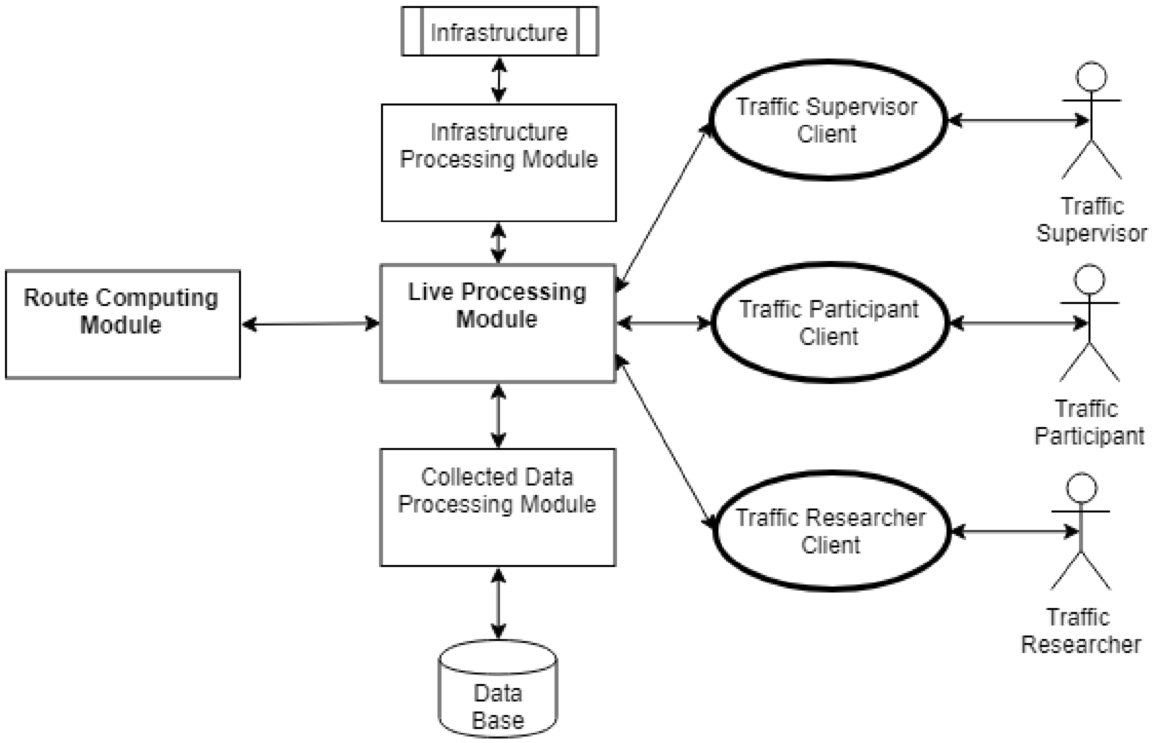

Infrastructure
Infrastructure Processing Module
Traffic Supervisor Client
Traffic Supervisor
Route Computing Module
Live Processing Module
Traffic Participant Client
Traffic Participant
Collected Data Processing Module
Traffic Researcher Client
Traffic Researcher
Data Base

Figure 4-2 Example of Architecture Based on the Conceptual Architecture

Figure 4.2. describes a system built on the conceptual architecture presented in the previous pages (4.3). The system above respects the structure proposed and has the main components and 5 other modules (3 Action Based Modules and 2 Query Based Modules).

- **Live Processing Module**

As described at point 4.2.1., the Live Processing Module is the core module of the system. It holds the current traffic status, updates it constantly and communicates with all the other modules of the system.

The Live Processing Module has 2 type of inputs. The first type of input is traffic status updates, received from the Traffic Participant Client Action Based Module, Traffic Manager Client Action Based and, on request, from the 2 Query Based Modules of the system. The Live Processing Module uses these updates to keep the traffic status up to date.

The second type of input received are requests. The Traffic Participant Client can request routes and the Live Processing Server forwards the request to the Route Computing Module and after it receives the response it outputs that response to the Traffic Participant client. On the other hand, the Traffic Researcher Client can request for statistics on different topics of the traffic status. When that input is received, the Live Processing Server computes and outputs the requested statistics.

- **The Route Computing Module**, as described at point 4.2.2., has the purpose to answer route requests with a route, based on the current status of the traffic, if it is possible to build one, or with an error message if the route cannot be computed.

- **Action Based Modules**
  - **The Traffic Participant Client** Module is the client application that can be used by the Traffic Participant stakeholder. This module aids the Traffic Participant in the process of requesting a route.

  After a route is received the module presents the Traffic Participant with the route, guides them and sends constant location updates to the Live Processing Module, so the traffic status can be updated.

  - **The Traffic Manager Client** Module is the client application that can be used by the Traffic Manager stakeholder. It offers the Traffic Manager the possibility to bring changes to the traffic context, like blocking a certain area of the city for various events.

  - **The Traffic Researcher Client** Module is the client application that can be used by the Traffic Researcher stakeholder. It offers the Traffic Researcher the possibility to request the Live Processing Module for various statistics on what concerns the traffic.

- **Query Based Modules**
  - **The Collected Data Processing Module** is the Query Based Module that is responsible with storing the traffic data received or computed by the Live Traffic Server. This module has direct connection to a database in which it stores the collected data, and from which it retrieves and analyses traffic data. It can provide the

    Live Traffic Module, on demand, with information regarding the overall usual status of the traffic at certain times/dates in the past.

    - **The Infrastructure Processing Module** is the Query Based Module that is responsible for communicating with the infrastructure systems, like live traffic cameras or proximity car counters, and build a status traffic model based on the information received from these systems. It is able to answer the Live Traffic Modules requests with the live status traffic computed using the traffic data from the infrastructure.

- A route request Use-Case, from the perspective of the Live Traffic Module, is presented in the following flow diagram.

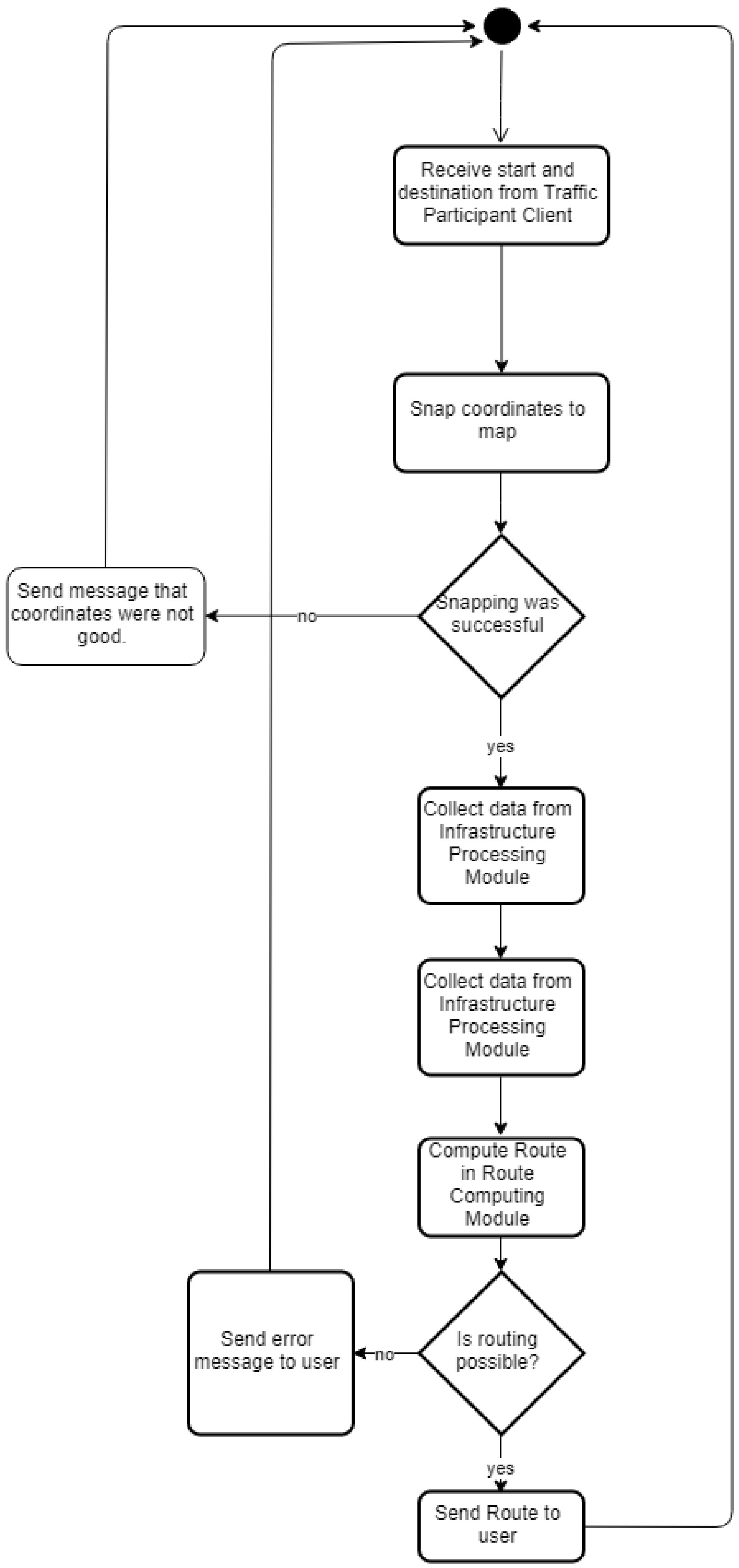

Receive start and destination from Traffic Participant Client
Snap coordinates to map
Snapping was successful
no
Send message that coordinates were not good.
yes
Collect data from Infrastructure Processing Module
Collect data from Infrastructure Processing Module
Compute Route in Route Computing Module
Is routing possible?
no
Send error message to user
yes
Send Route to user

Figure 4-3 Flow Diagram of Route Request Use-Case

4.3. TCS-3C - Design

4.3.1. Design

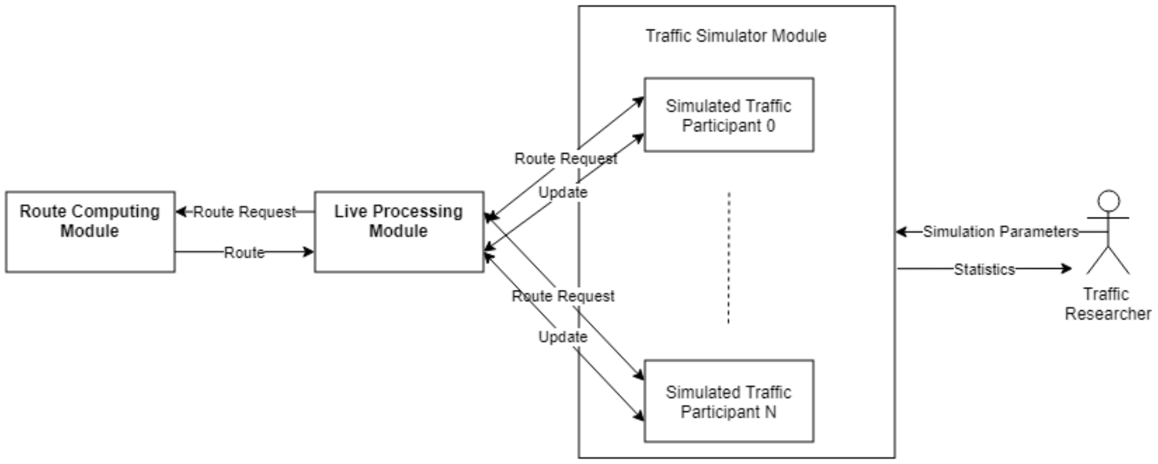


Figure 4-4 Design of TCS-3C

Figure 4.4. describes the architecture of TCS-3C. It respects the conceptual diagram described earlier (4.2.). It has the main modules, the Live Traffic Module and the Route Computing Module, and one Action Based Module, the Traffic Simulator Module.

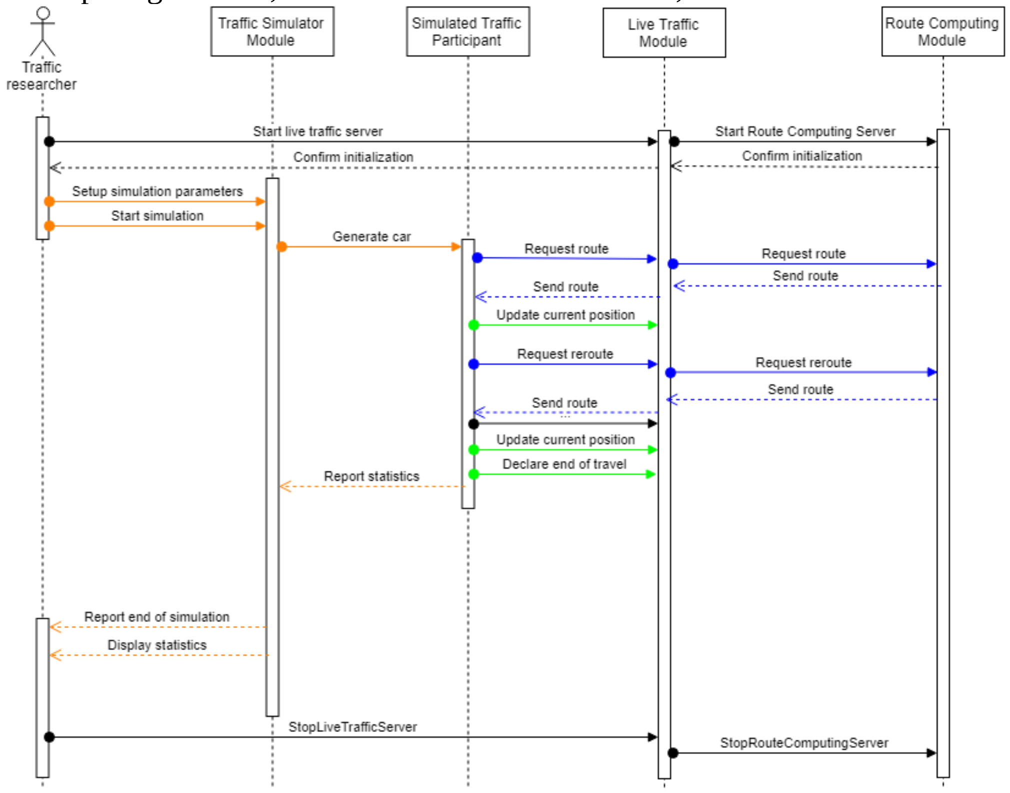


Figure 4-5 Sequence Diagram of a TCS-3C

Figure 4.5. describes a simulation in terms of inter-module communication in the TCS-3C system.

- **The Live Processing Module**, as it was previously described, is the main module of the system. It has the purpose of connecting all other modules in the system. In this case, the Live Processing Module has a direct connection to the Route Computing Module and to the Traffic Simulator Module.

This module manages the request for routes received from the Simulator Module. Whenever a route request is received, the Live Traffic Module forwards it to the Route Computing Module, together with the current status of the traffic. It then expects a route and forwards it to the simulated traffic participant. These actions can be found colored blue in Fig 4.4.

It maintains the current status of the traffic by updating it periodically when location updates are received from traffic participants simulated by the Traffic simulator. These actions can be seen in color Green in figure 4.4.

- **The Route Computing Module**, as described at point 4.2.2., has the purpose to answer route requests with a route, based on the current status of the traffic, if it is possible to build one, or with an error message if the route cannot be computed.

- **The Traffic Simulator Module** is the interface of the Traffic Researcher in TCS-3C. Its purpose is to generate a traffic context based on a set of parameters and create statistics on the effects of Traffic Participants choices, made in traffic. These effects can be studied by varying a threshold (0%-100%) of people taking the best route over the shortest one and comparing the total times of the simulated contexts.

It simulates traffic participants, giving each one of them a type of behavior, a starting location and an ending location. Each one of these traffic participants requests a route and starts following it, according to their behavior, updating their position and requesting reroutes when necessary.

During the simulation process, the module gathers statistics from the simulated traffic participants and displays them to the user at different points in the simulation and at the end of it.

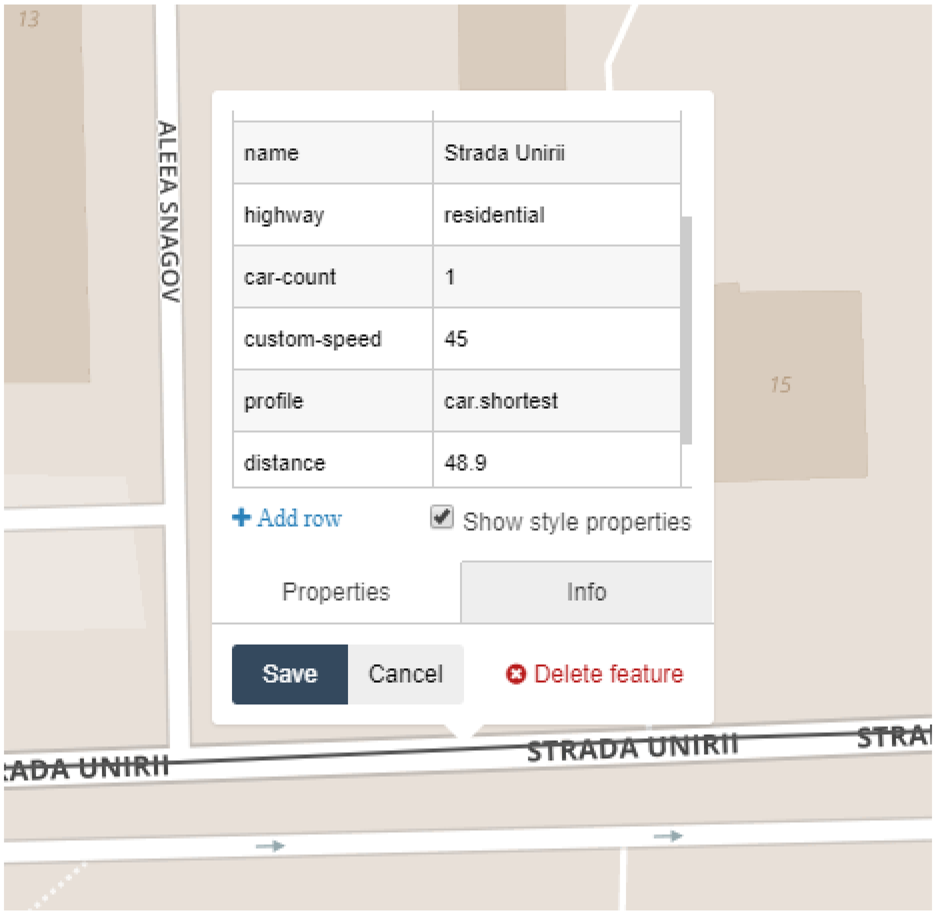
These actions can be seen in Fig 4.4. colored orange.

4.3.2. Traffic Data

Previous points stated that the Live Traffic Module holds the traffic status in order to be used when the routing is done. This traffic status is a combination of the already existent map information and the traffic data collected by the Live Traffic Module. Besides the basic data related to the maps, presented at 3.2, in order to hold the traffic related data, each edge has two additional characteristics:

- car-count: The number of cars that can be found on that edge at the time of the query
- custom-speed: A speed that was computed for that edge

An example can be found in the following image.



name	Strada Unirii
highway	residential
car-count	1
custom-speed	45
profile	car.shortest
distance	48.9

Figure 4-6 Screenshot of Partial Information Recorded about an Rdge on the Map

Normally, the speed associated with an edge is hardcoded and is equal to the speed limit legally imposed for that edge. In the case of TCS-3C, besides the legal speed limit, the number of cars present at one time on that edge and their average length influences the speed that can be reached on that edge. The function that determines the new speed can be easily changed in the system and the current form of this function is the following:

$$custom\_speed = legal\_speed * (1 - \frac{car\_count * average\_car\_length}{edge\_distance})$$

For example, in the picture above the legal speed limit is 50 km/h in that area, the number of cars on that edge is 1, the average car length is 4 meters and the edge distance is 48.9 meters. That gives us, using the formula above, a custom speed of 45.7 km/h, which is rounded to 45.

This function dictates the new speed for every edge that will be used for routing.

4.3.3. Routing process

The routing process takes place in the Route Computing Module.

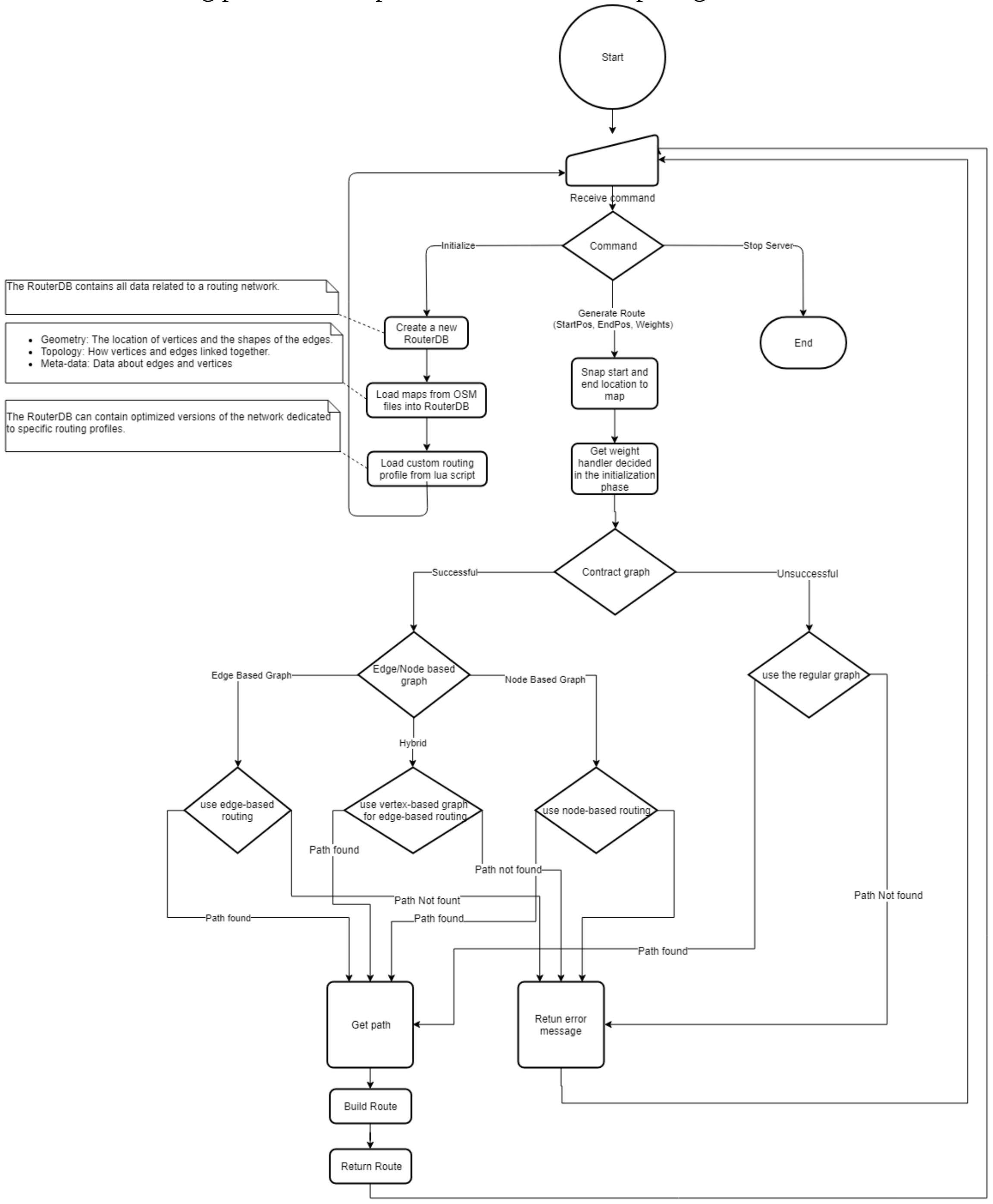

Start
Receive command
Command
Initialize
Stop Server
End
Generate Route (StartPos, EndPos, Weights)
The RouterDB contains all data related to a routing network.
Geometry: The location of vertices and the shapes of the edges.
Topology: How vertices and edges linked together.
Meta-data: Data about edges and vertices
The RouterDB can contain optimized versions of the network dedicated to specific routing profiles.
Create a new RouterDB
Load maps from OSM files into RouterDB
Load custom routing profile from lua script
Snap start and end location to map
Get weight handler decided in the initialization phase
Contract graph
Successful
Unsuccessful
Edge/Node based graph
Edge Based Graph
Node Based Graph
Hybrid
use the regular graph
use edge-based routing
use vertex-based graph for edge-based routing
use node-based routing
Path found
Path not found
Path Not fount
Path found
Path found
Path found
Path Not found
Get path
Retun error message
Build Route
Return Route

Figure 4-7 Flowchart for the Routing Process.

The first steps of the process include deciding whether the traffic status should be reloaded or not. If there were enough changes in the traffic context, the routerDb is reloaded.

The second part of the routing process is validating the start and end locations. It consists of checks whether the two pairs of coordinates are included in the map that is currently concerned. If the points are inside the map, the next step is checking if these points can be snapped to the road network in the map or if they are too far.

After the input data is validated, the next step is trying to contract the graph. This step minimizes the size of the graph so less effort is required in the route computing process. This process removes from the routerDb the information that is not used for the current routing profile. For example, if the current routing is only done for cars, all information about pedestrians and bicycles can be removed from the routerDb, as it won't be used. This step also removes the nodes that have under three neighbors. The contraction also includes transforming the basic graph into an edge-based graph, if possible. This action would make the edges the main point of interest in the routerDb and the nodes would become edges. This would allow speed up in the routing process and would make it easier to model turn restrictions.

The last part of the process is the actual routing. Route computing is done using a custom bidirectional Dijkstra's algorithm, for each form of the graph. This algorithm There are 2 types of routing in TCS-3C:

- shortest - a bidirectional Dijkstra's algorithm adaptation that takes as weights the length of the edges and the legal speed limit. This type of routing is used by the greedy Traffic Participants.
- best - a bidirectional Dijkstra's algorithm adaptation that takes as weights the length of the edges and the custom speed computed for each edge (4.3.2)

Chapter 5. Detailed Design and Implementation

5.1. Components Architecture

In terms of implementation, the TCS-3C is built using Microsoft's technical stack. The Live Traffic Server and the Route Computing Server are both .NET Core 2.1 APIs, while the Traffic Simulator is a Console Application in .NET Framework 4.6.1. In this chapter each one of these elements will be discussed in terms of implementation and the communication between these modules.

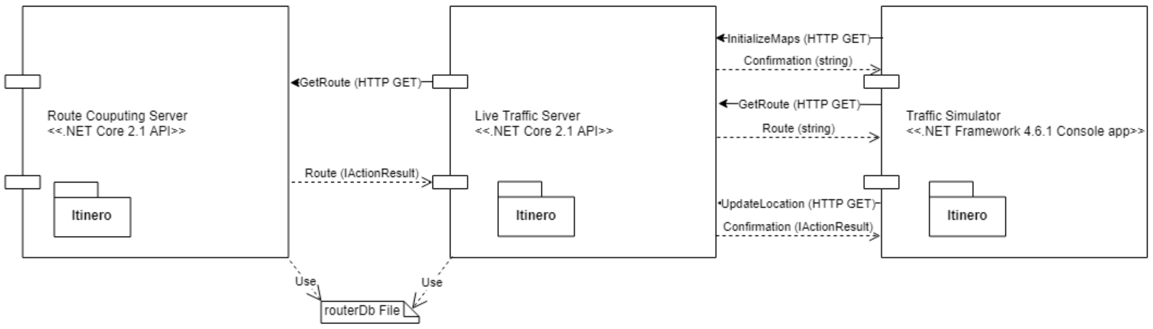


Figure 5-1 High Level design of TCS-3C

In Figure 5-1, a high level design of the system is described. At first sight, it is visible that all the modules in the system have the Itinero package. This package helps keep the datastructures similar over the entire system.

In terms of communication, as discussed at point 3.5, the TCS-3C presents an inter module communication based on the HTTPS protocol. Each one of the main components exposes several endpoints that are accessed via HTTP requests. The only exception concerning the communication is the way the Live Traffic Server communicates the map status: it is done using the routerDb file.

5.2. Live Traffic Server

5.2.1. Design

The Live Traffic Server is a .NET Core 2.1 API that, as described at point 4.3 is the core server of the application.

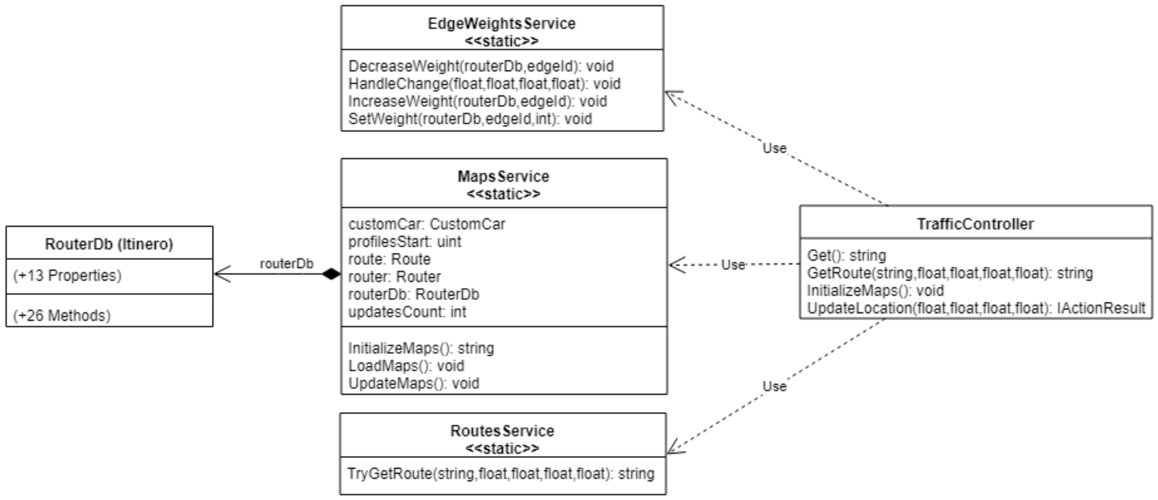


Figure 5-2 High Level Class Diagram of Live Traffic Server

Figure 5-2 presents the high level class diagram of the Live Traffic Server. For the current version of TCS-3C the high level architecture is rather simple. It contains four main classes:

- **Traffic Controller** is an extension of the ControllerBase class in .NET Core. It is responsible with exposing the endpoints and handling the requests. There are four endpoints exposed in the Traffic Controller:

  The first one is a test GET endpoint that returns „Live Traffic Server is Running. Waiting for commands.” if the server is running.

  The second one is the InitializeMaps GET endpoint which calls the MapsService.InitializeMaps() method. It returns a confirmation string.

  The third one is the GetRoute endpoint which receives five parameters: the routing profile (shortest or best), latitude and longitude of the start and end location. This endpoint delivers the resonsability to the RoutesService by calling the TryGetRoute method in the service with the received parameters. It returns the route received from the RoutesService as a string.

  The fourth endpoint is UpdateLocation. It receives as parameters the coordinates of the middle of the edge a Traffic Participant is currently leaving and the coordinates of the middle of the edge the participant is moving to. It calls the EdgeWeightsService.HandleChange method with the received parameters. It returns an OK(„succesful”) ActionResult if the update was succesful of a BadRequest(errorMessage) otherwise.

- **The Maps Service** is a static service that is responsible with maintaining the current status of the maps and handling all the actions performed with it.

  The first static member is the routerDb. This object holds the entire map and profiles associated with each edge. It is initialized in the LoadMaps or in the InitializeMaps method.

The second static member is the customCar DynamicVehicle. It is used for snapping points to the map and contracting of the graph (4.3.3). This is loaded in the InitializeMaps method from the custom .lua script.

The third static member is the router. This member chaches the costomCar profile and it is used for snaping points to the map.

The profilesStart static member holds the index where the custom edge profiles start.

The updateCount static member holds the number of updates done. This member is used for the Live Traffic server to know when to Update the maps in the routerDb file and flag the Route Computing server that it should reload the maps aswell. The maps are updated and this member resets when the UpdatesNecesaryForMapRefresh parameter in Constants.cs is smaller than the number of updates.

The first method is LoadMaps. It is used to load the routerDb from a file.

The second static method is the InitializeMaps method. It loads a new routerDb directly from the pbf format of the map. On the loaded map it adds 500 new custom edge profiles. One for each combination of car-count (0-99) and custom-speed (1-50). It initializes the customCar routing profile, caches it into the router and updates the routerDb file, by calling the UpdateMaps method.

Figure A1-1 presents the code described in the previos paragraphs.

The last static method is the UpdateMaps. It is used to serialize the routerDb and write it into the routerDb file.

- **The RoutesService** static class is has the responsability to handle the routing requests. It has only one static asyncronous method TryGetRoute. This method receives as parameters form the TrafficController a routing profile name („shortest" or „best") and the coordinates of the start and end point of the request.

  Firstly, if the updatesCount in the MapsService is greater or equal than the UpdatesNecesaryForMapRefresh parameter in Constants.cs, The Routes Service calls the UpdateMaps method of the MapsService and flags that the following route request to the Route Computing Server should notify the changing of the routerDb file.

  Then, it builds a HTTP request and calls the Route Computing Server for a route. In the case of a BadRequest it updates the maps again and tries a new call by flagging the Route Computing Server to refresh the maps it has. The last step is returning the response received from the Route Computing Server to the TrafficContriller. The code can be seen below.

- **The EdgeWeightsService** static class is responsible with managing all the actions taken in the map at edge level. This is where the status of the traffic is updated by „moving" cars through the virtual edges.

Figure 5-2 presents the three static methods of the EdgeWeightsService. A change in the traffic context is the moment when a Traffic Participant goes from one edge to another. To deonte that, the TrafficControllers' UpdateLocation endpoint is called with two coordinates: the middle of the edge the Traffic Participant is leaving and the middle of the edge the Traffic Participant is joining. The TrafficController calls the HandleChange static method of the EdgeWeightsService.

The HandleChange static method takes as arguments two sets of coordinates (previousEdgeLon, previousEdgeLat) and (currentEdgeLon, currentEdgeLat). If the previous coordinate is (0,0) that means a new traffic participant is joining the traffic. If the previous coordinate is not (0,0) but the current one is, that means a traffic participant just left the traffic. If none of the 2 coordinates are (0,0) that denotes only the action of an already existing traffic participant. The HandleChange method snaps the nonzero coordinates to the map and gets the Id of the edges taken into consideration. The next step is decreasing the weight of the previous weight, as a car just left that edge, and increasing the weight of the current edge. This is done using the DecreaseWeight and IncreaseWeight static methods.

The IncreaseWeight and DecreaseWeight work in a similar manner. They receive as parameters the routerDb and an edge Id. First they retreive the profile of the edge. If this profiles Id is lower than the profilesStart variable in the MapsService it means that is hasn't been visited before and it is assigned the first custom edge profile (car-count =0, custom-speed=0). The next step is to compute the new number of cars, by retreiving the car-count value and increasing/decreasing it. By aplying the formula described at 4.3.2, a new custom-speed is computed. Using the new car-count and custom-speed, the Id of the new edge profile is computed and assigned to the edge.

The code of the IncreaseWeight method can be seen in A1-3 as a screenshot from Visual Studio.

5.2.2. Flow

The live Traffic Server has three main flows, associated with the endpoints exposed by the Traffic Controller: InitializeMaps, UpdateLocation and GetRoute. These flows take different code paths through the previously described Classes and methods in order to return an answer.

- **The InitializeMaps flow**

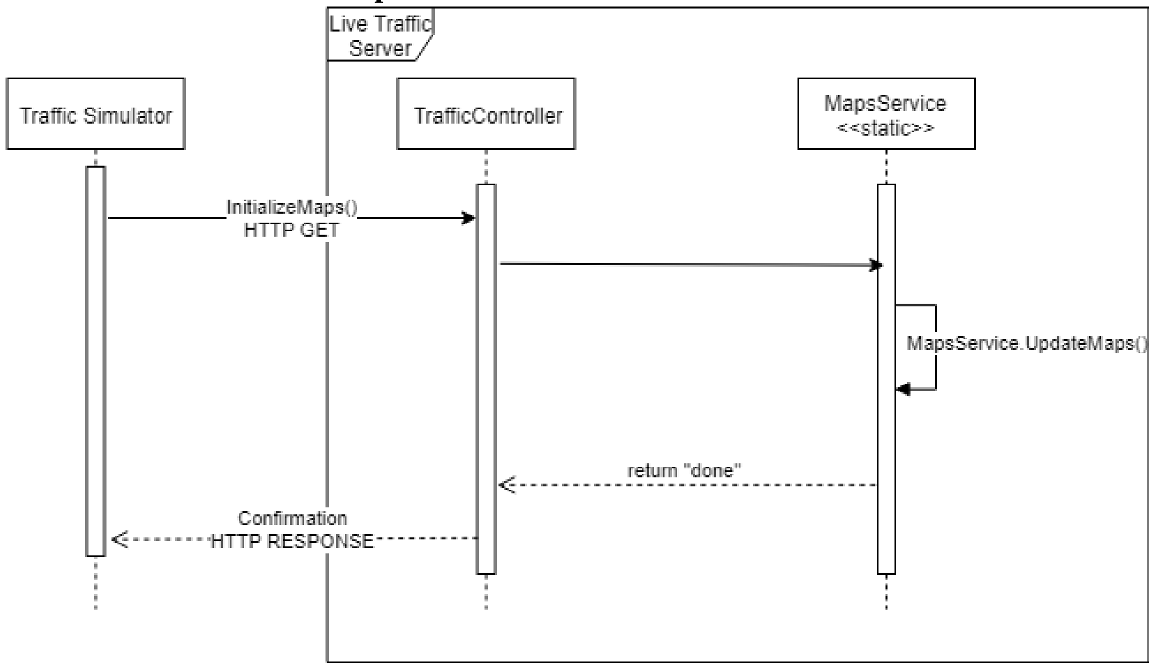


Figure 5-3 Flow Diagram of the InitializeMaps use-case in the Live Traffic Server

In Figure 5-3 the code flow of the InitializeMaps use-case is described. As stated earlier, the command is received via HTTP GET by the TrafficController. The controller delivers the responsability to the MapsService which handles the command as described at point 5.2.1.

- **The GetRoute flow**

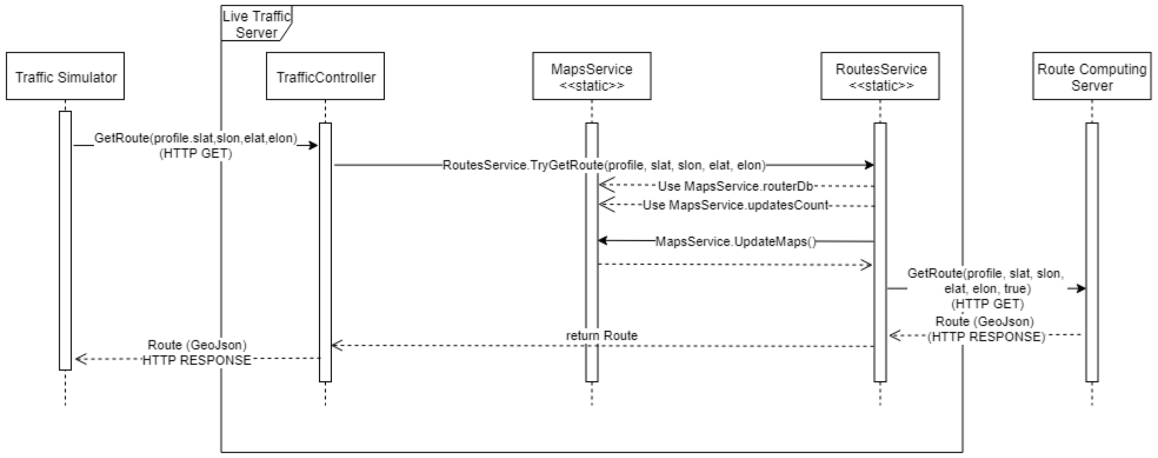

Live Traffic Server
Traffic Simulator
TrafficController
MapsService <<static>>
RoutesService <<static>>
Route Computing Server
GetRoute(profile.slat,slon,elat,elon) (HTTP GET)
RoutesService.TryGetRoute(profile, slat, slon, elat, elon)
Use MapsService.routerDb
Use MapsService.updatesCount
MapsService.UpdateMaps()
GetRoute(profile, slat, slon, elat, elon, true) (HTTP GET)
Route (GeoJson) (HTTP RESPONSE)
return Route
Route (GeoJson) HTTP RESPONSE

Figure 5-4 Flow Diagram of the GetRoute use-case in the Live Traffic Server

In Figure 5-4 the code flow of the GetRoute use-case is described. As stated in 5.2.1, the command is received by the TrafficController via HTTP protocol. The controller delivers responsability to the RoutesService by using the TryGetRoute method. The service uses the routerDb and updatesCount variables from the MapsService and, if necesarry, it updates the routerDb file using the UpdateMaps method in the MapsService. The server then calls the Route Computing server via HTTP GET and returns the response received from it to the controller.

- **The UpdateLocation flow**

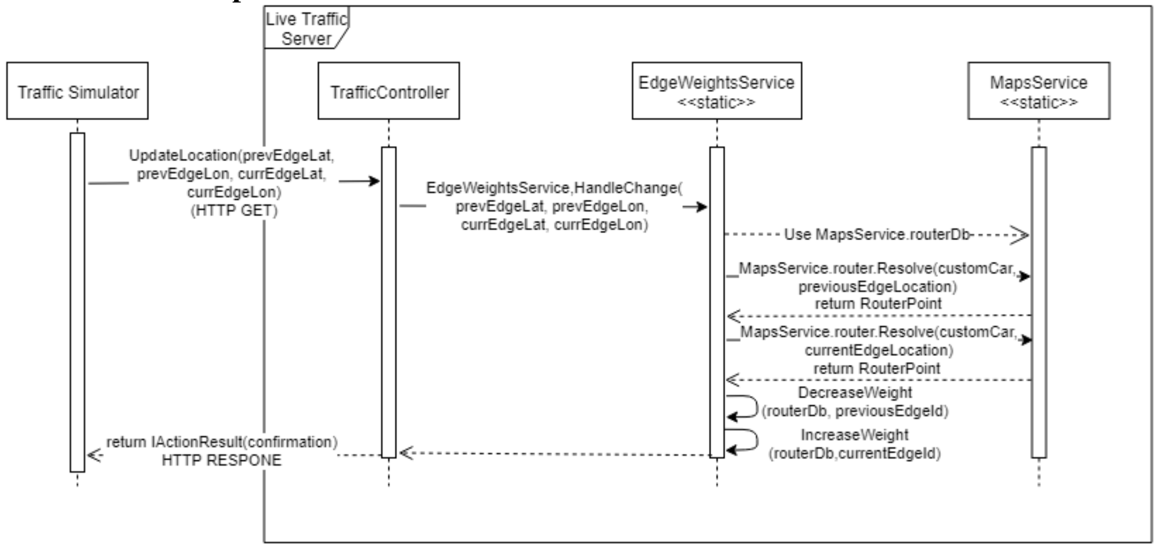

Live Traffic Server
Traffic Simulator
TrafficController
EdgeWeightsService <<static>>
MapsService <<static>>
UpdateLocation(prevEdgeLat, prevEdgeLon, currEdgeLat, currEdgeLon) (HTTP GET)
EdgeWeightsService.HandleChange(prevEdgeLat, prevEdgeLon, currEdgeLat, currEdgeLon)
Use MapsService.routerDb
MapsService.router.Resolve(customCar, previousEdgeLocation)
return RouterPoint
MapsService.router.Resolve(customCar, currentEdgeLocation)
return RouterPoint
DecreaseWeight (routerDb, previousEdgeId)
IncreaseWeight (routerDb,currentEdgeId)
return IActionResult(confirmation) HTTP RESPONE

Figure 5-5 Flow Diagram of UpdateLocation use-case in the Live Traffic Server

In Figure 5-5 the code flow of the UpdateLocation use-case is described. As stated earlier, the command is received via HTTP GET by the TrafficController. The controller delivers the responsability to the EdgeWeightsService, by calling he HandleChange method. Using the routerDb and the router from the MapsService, the EdgeWeightsService snaps the input coordinates to the map, decreases the weight of the previous edge and increases the weight of the current edge.

5.3. Route Computing Server

5.3.1. Design

The Route Computing Server is a .NET Core 2.1 API that, as described at point 4.2.2, is the component that is responsible with managing routing requests from the Live Traffic Server.

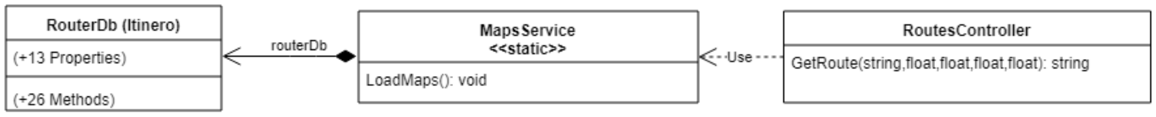


Figure 5-6 High level Class Diagram of Route Computing Server

In Figure 5-6 the Route Computing Server is presented from a high level perspective. This diagram does not include the class structure behind the RouterDb class, as this diagram presents only the API project. That makes the high level class diagram rather simple.

The server contains 2 main classes: the controller called RoutesController and the static service that handles the maps, called MapsService.

- **The RoutesController** class is an extension of the ControllerBase class in .NET Core. It is responsible with exposing the endpoints and handling the requests. The definition of this function can be found in Figure 5-7. There are two endpoints exposed by the RoutesController.

  The first endpoint, Get(), is a test endpoint. It returns the message "Route Computing Server is Running. Waiting for commands." if the server is running.

  The second endpoint, GetRoute, is responsible with receiving and handling route requests. It receives as parameters a routing profile, “shortest” or “best”, four float values that represent the coordinates of the start and end locations of the requested route and a flag announcing the server whether it should refresh its current map status or it should route using the currently loaded one.

Figure A1-4 presents the code of the GetRoute endpoint in a screenshot from Visual Studio.

The first thing done is checking wether the mapRefresh flag is active or not. If it is, the LoadMaps method from the MapsService is called, in order to reload the maps.

The second step is computing the route. Based on the chosen routing profile, the controller calls the Calculate method of the router member in the MapsService and sends as parameters the routing profile chosen by the caller, and the two coordinates for the starting and ending point of the route. It then serializes the received route into a GeaoJson format end returns it to the caller with status OK.

All these steps are surrounded with a try/catch clause, so in case any of the steps fail, the error message is returned to the caller with status BadRequest.

- **The Maps Service** is a static service that is responsible with loading and maintaining the current status of the maps. The definition of this function can be found in Figure 5-7. The Maps service has three static members and a static method:
  - **The routerDb** static member, of type RouterDb, described in [20],is the member that holds all the map and traffic information. It is loaded at the start of the server and is updated constantly using the LoadMaps method.
  - **The router** static member, of type Router, described in [21], is the member that is used for routing in the routerDb. It is reinstantiated every time the maps are updated in the LoadMaps static method.
  - **The customCar** static member, of type DynamicVehicle, described in [22], is the member that defines the profiles of the routings done in the routerDb. This is loaded from a custom Lua script.
  - **The LoadMaps** static method is responsible with refreshing the current routerDb by loading the new sattus from the routerDb file.
- **All the other functionalities** concerning snaping and routing are implemented in Itinero and are described in [23].

5.3.2. Flow

The Route Computing Server has one main flow: the GetRoute flow. This flow Starts when a route request is received by the RoutesController. An overview of the flow can be seen in Figure 5.16.

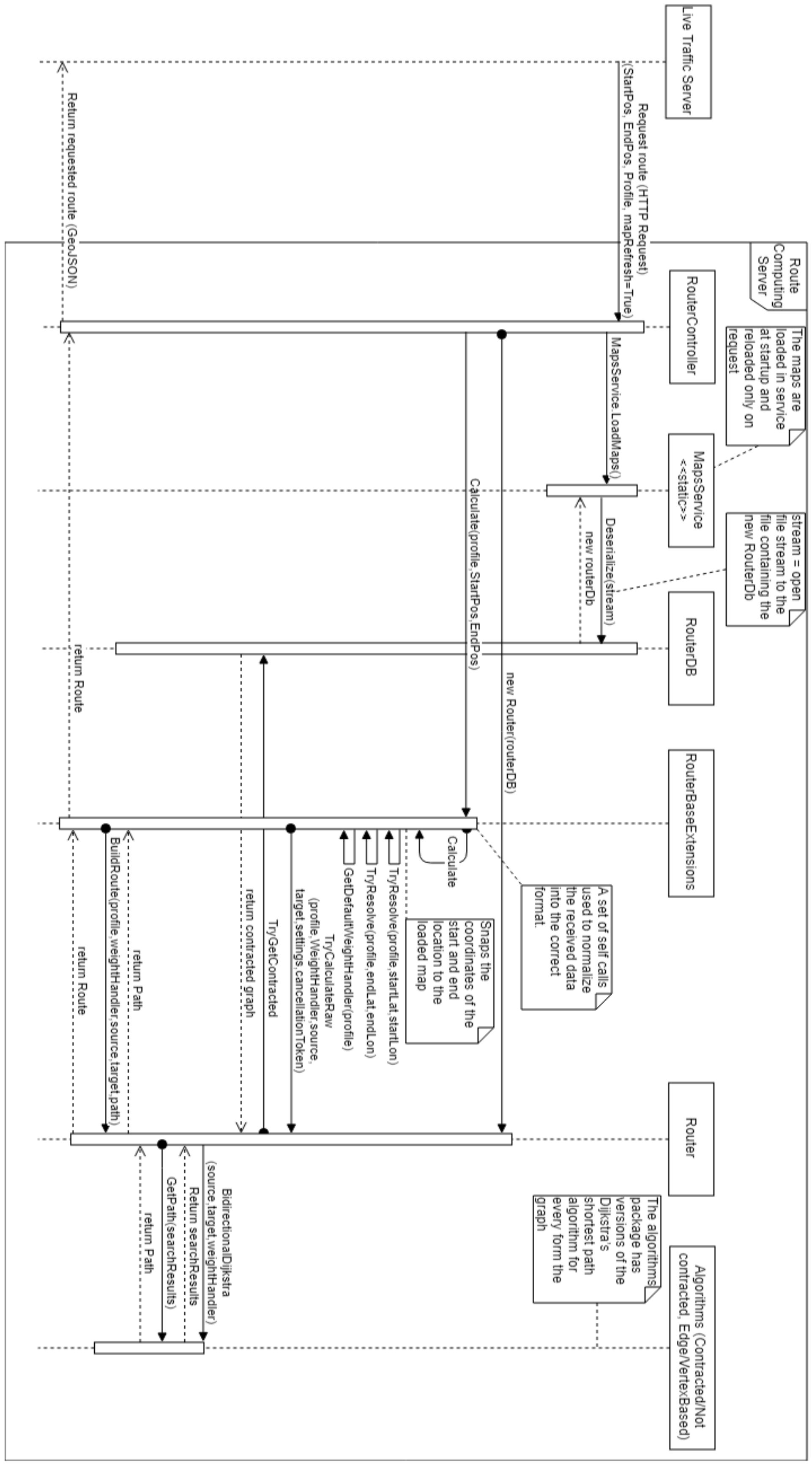

Live Traffic Server
Route Computing Server
RouterController
MapsService <<static>>
RouterDB
RouterBaseExtensions
Router
Algorithms (Contracted/Not contracted, Edge/VertexBased)
Request route (HTTP Request) (StartPos, EndPos, Profile, mapRefresh=True)
Return requested route (GeoJSON)
The maps are loaded in service at startup and reloaded only on request
MapsService.LoadMaps()
stream = open file stream to the file containing the new RouterDb
Deserialize(stream)
new routerDb
Calculate(profile,StartPos,EndPos)
new Router(routerDB)
return Route
A set of self calls used to normalize the received data into the correct format.
Calculate
TryResolve(profile,startLat,startLon)
TryResolve(profile,endLat,endLon)
GetDefaultWeightHandler(profile)
Snaps the coordinates of the start and end location to the loaded map
TryCalculateRaw (profile,WeightHandler,source, target,settings,cancellationToken)
TryGetContracted
return contracted graph
return Path
BuildRoute(profile,weightHandler,source,target,path)
return Route
The algorithms package has versions of the Dijkstra's shortest path algorithm for every form the graph
BidirectionalDijkstra (source,target,weightHandler)
Return searchResults
GetPath(searchResults)
return Path

Figure 5-7 Flow diagram of a Route Request with Map Refresh

Figure 5-7 is a flow diagram of the code path in the Route Computing Server for a route request. In the use case described the mapRefresh flag in the request is set to true, so the maps have to be reloaded from the routerDb file before starting the route computation.

5.4. Traffic Simulator

5.4.1. Design

The Traffic Simulator is a .NET Framework 4.6.1 Console Application that is responsible with creating a virtual traffic context. The simulator creates virtual Traffic Participants that mimic the actions real Traffic Participants: they request routes, update their positions and ask for reroutes periodically.

The Traffic Simulator also gathers statistics from the traffic participants and processes them in order to output System and Simulation Statistics (6.4).

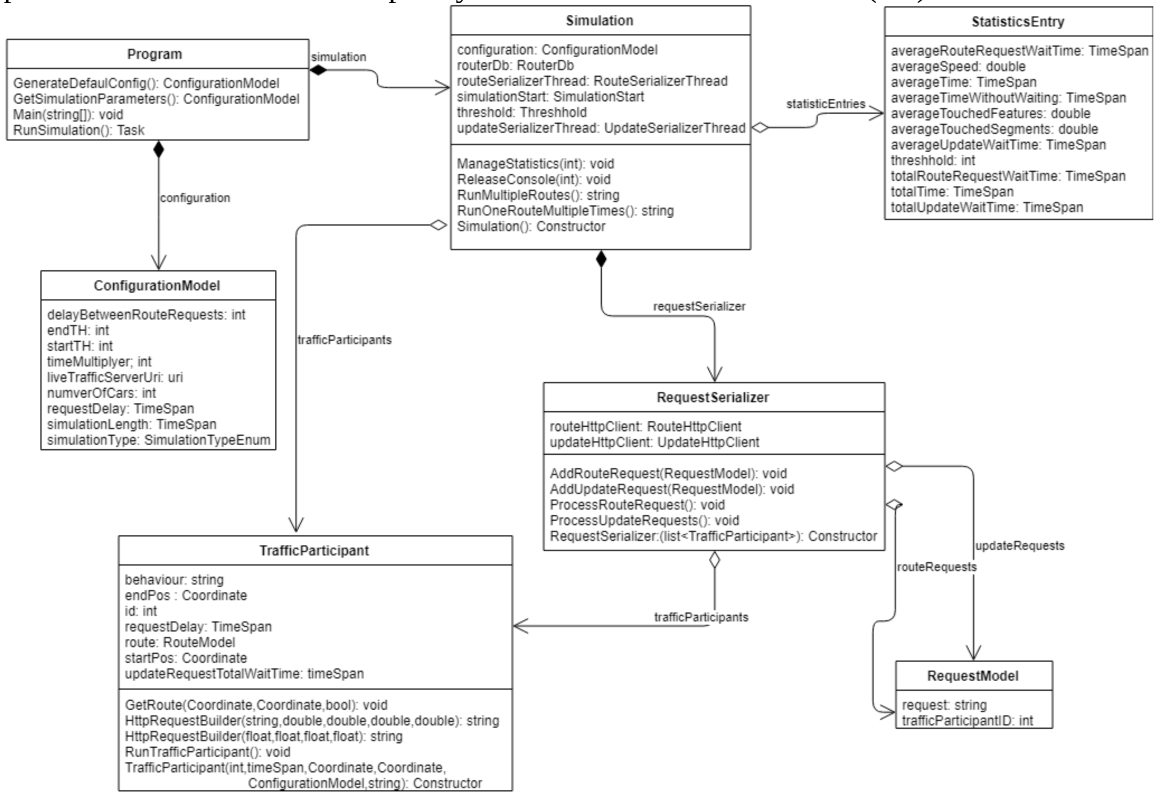

Program
GenerateDefaulConfig(): ConfigurationModel
GetSimulationParameters(): ConfigurationModel
Main(string[]): void
RunSimulation(): Task
simulation
Simulation
configuration: ConfigurationModel
routerDb: RouterDb
routeSerializerThread: RouteSerializerThread
simulationStart: SimulationStart
threshold: Threshhold
updateSerializerThread: UpdateSerializerThread
ManageStatistics(int): void
ReleaseConsole(int): void
RunMultipleRoutes(): string
RunOneRouteMultipleTimes(): string
Simulation(): Constructor
statisticEntries
StatisticsEntry
averageRouteRequestWaitTime: TimeSpan
averageSpeed: double
averageTime: TimeSpan
averageTimeWithoutWaiting: TimeSpan
averageTouchedFeatures: double
averageTouchedSegments: double
averageUpdateWaitTime: TimeSpan
threshhold: int
totalRouteRequestWaitTime: TimeSpan
totalTime: TimeSpan
totalUpdateWaitTime: TimeSpan
configuration
ConfigurationModel
delayBetweenRouteRequests: int
endTH: int
startTH: int
timeMultiplyer; int
liveTrafficServerUri: uri
numverOfCars: int
requestDelay: TimeSpan
simulationLength: TimeSpan
simulationType: SimulationTypeEnum
trafficParticipants
requestSerializer
RequestSerializer
routeHttpClient: RouteHttpClient
updateHttpClient: UpdateHttpClient
AddRouteRequest(RequestModel): void
AddUpdateRequest(RequestModel): void
ProcessRouteRequest(): void
ProcessUpdateRequests(): void
RequestSerializer:(list<TrafficParticipant>): Constructor
updateRequests
routeRequests
trafficParticipants
TrafficParticipant
behaviour: string
endPos : Coordinate
id: int
requestDelay: TimeSpan
route: RouteModel
startPos: Coordinate
updateRequestTotalWaitTime: timeSpan
GetRoute(Coordinate,Coordinate,bool): void
HttpRequestBuilder(string,double,double,double,double): string
HttpRequestBuilder(float,float,float,float): string
RunTrafficParticipant(): void
TrafficParticipant(int,timeSpan,Coordinate,Coordinate,
ConfigurationModel,string): Constructor
RequestModel
request: string
trafficParticipantID: int

Figure 5-8 High level class diagram of Traffic Simulator

In Figure 5-8, the Traffic simulator is described using a high level class diagram. The simulator is composed of four main classes and uses 4 types of models.

- **The Program class** is the first class that is loaded when the simulator is started.

The definition of this class can be found in Figure 5-8. The class contains two static members and three static methods.

  - simulation – instance of the Simulation class that represents a simulation with the parameters fond in the configuration

    - configuration – instance of the ConfigurationModel class that holds all the information about the curently run simulation
    - GenerateDefaulConfigs – method that generates a basic configurations file
    - GetSimulationParameters – method that reads the simulation parameters from the SimulationParameters.json file and popultes the configuration field
    - Main – method that is run when the project is built
    - SunSimulation – method that instantiates the simulation parameter and runs the proper kind of simulation based on the simulation parameters

- **The Simulation class** contains all the members and methods necessary for a simulation. It also contains separate methods that define the flow for every type of simulation. As Figure 5-8 shows, the class contains 10 fields and 7 methods. The most important ones are:
    - trafficParticipants – list of TrafficParticipant instances that represent the simulated traffic participants
    - routeSerializerThread – instance of Thread class that represents the thread the RequestSerializer uses to process route requests
    - updateSerializerThread – instance of Thread class that represents the thread the RequestSerializer uses to process location updates
    - threshold – the percentage of people taking the „best” route over the „shortest” route
    - statistics – list of StatisticsEntryModel that holds the statistics for every completed threshold in the simulation
    - RunOneRouteMultipleTimes – method that implements the functionality of Scenario 1 (6.3)
    - RunMultipleRoutes – method that implements the functionality of Scenario 2 (6.3)

- **The TrafficParticipant class** defines the behaviour and features of a simulated Traffic Participant. This class includes 26 features and 12 methods. The most important ones are:
    - id – the unique identifier of the Traffic Participant
    - requestDelay – a TimeSpan that represents the momment in simultion this Traffic Participant requests a route
    - startPos – the coordinates of the start location of the route for this traffic participant
    - endPos – the coordinates of the end location of the route for this traffic participant
    - behaviour – string field that reflects whether the traffic participant is greedy or not when it comes to route choice
    - TrafficParticipant – constructor method

  - RunTrafficParticipant – method that describes the behaviour of the traffic participant in the simulation. Every instance of this class has the RunTrafficParticipant in a different thread.
  - GetRoute – method used by the Traffic Participant to add a route request in the RequestSerializer and await the response

- **The RequestSerializer class** is responsible with serializing the requests from traffic participants to the Live Traffic Server. This class sends the requests to the Live Traffic Server, one by one and awaits their response. It contains 12 fields and 5 methods, the most important ones being:
  - trafficParticipants – the list of all active Traffic Participants in the simulation
  - routeRequests – a list of RequestModels representing the not yet completed route requests
  - updateRequests – a list of RequestModels representing the not yet completed update requests
  - routeHttpClient – instance of HttpClient that is responssible with the communication with the GetRoute endpoint in the Live Traffic Server
  - updateHttpClient – instance of HttpClient that is responssible with the communication with
  - AddRouteRequest – method that is responsible with adding a new route request to the routeRequests list to be processed
  - AddUpdateRequest – method that is responsible with adding a new update request to the routeRequests list to be processed
  - ProcessRouteRequests – method responsible with serving the route requests from the routeRequests list. This method runs on a separate thread for the entire time of the simulation
  - ProcessUpdateRequests – method responsible with serving the update requests from the routeRequests list. This method runs on a separate thread for the entire time of the simulation

5.4.2. Flow

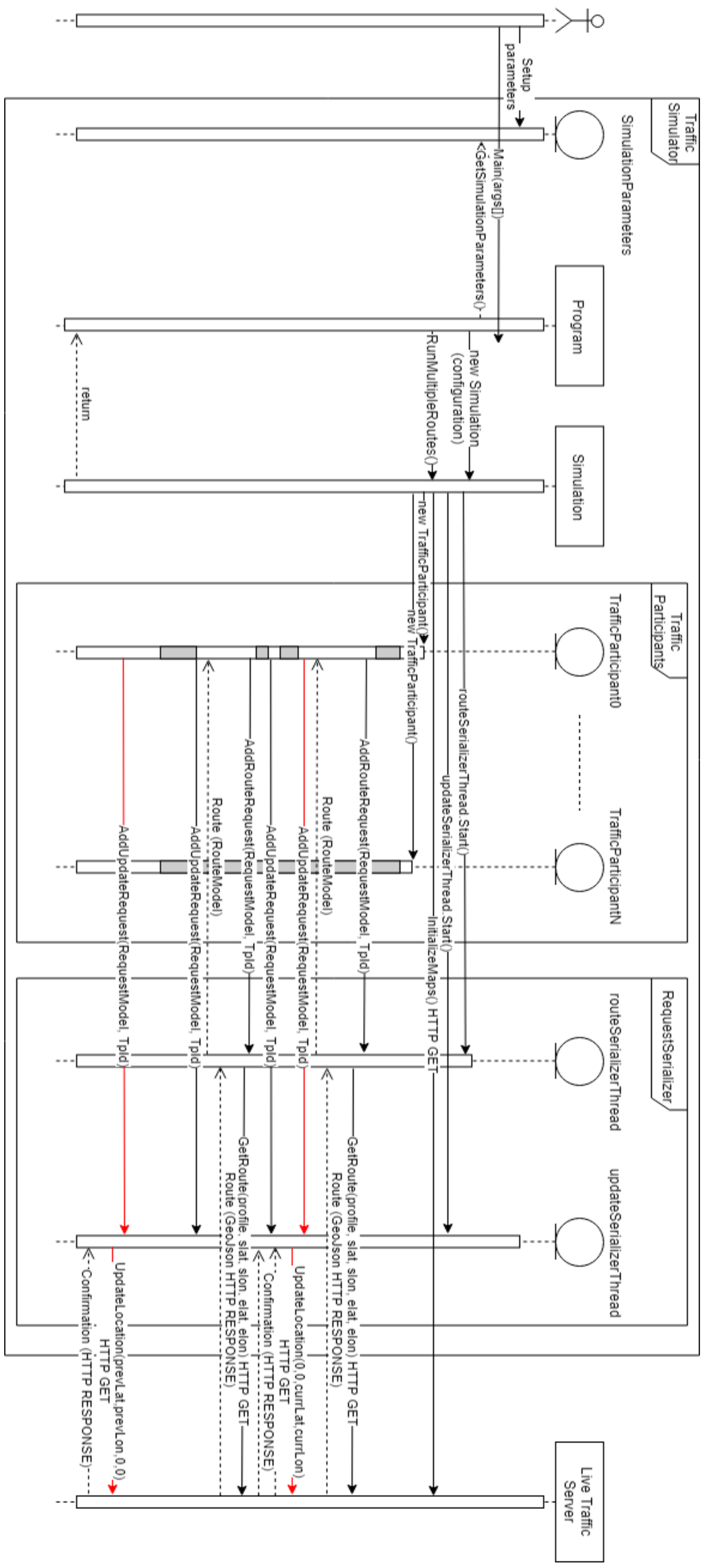

Setup parameters
Traffic Simulator
SimulationParameters
Program
Simulation
Main(args[])
<GetSimulationParameters()
new Simulation (configuration)
RunMultipleRoutes()
return
Traffic Participants
TrafficParticipant0
TrafficParticipantN
new TrafficParticipant()
new TrafficParticipant()
routeSerializerThread.Start()
updateSerializerThread.Start()
InitializeMaps() HTTP GET
AddRouteRequest(RequestModel, Tpid)
Route (RouteModel)
AddUpdateRequest(RequestModel, Tpid)
AddRouteRequest(RequestModel, Tpid)
AddUpdateRequest(RequestModel, Tpid)
Route (RouteModel)
AddUpdateRequest(RequestModel, Tpid)
AddUpdateRequest(RequestModel, Tpid)
RequestSerializer
routeSerializerThread
updateSerializerThread
GetRoute(profile, slat, slon, elat, elon) HTTP GET
Route (GeoJson HTTP RESPONSE)
UpdateLocation(0,0,currLat,currLon) HTTP GET
Confirmation (HTTP RESPONSE)
GetRoute(profile, slat, slon, elat, elon) HTTP GET
Route (GeoJson HTTP RESPONSE)
UpdateLocation(prevLat,prevLon,0,0) HTTP GET
Confirmation (HTTP RESPONSE)
Live Traffic Server

Figure 5-9 Sequence Diagram of the Traffic Simulator code

In Figure 5-9 the flow of a simulation is described in the form of a sequence Diagram. A simulation starts with the Traffic Researcher setting up the parameters for a simulation:

- The type of the simulation (Scenario 1 (6.3) or Scenario 2 (6.3))
- Number of Traffic Participants in the simulation
- The delay between two Traffic Participants joining the simulation
- The start threshold of the simulation
- The end threshold of the simulation
- The time multyplier of the simulation (how many times slower is the simulation compared to real time)
- The URL for the Live Traffic Server

The Traffic Researcher starts the simulator. The Simulator reads the simulation parameters in the configuration field and creates a new Simulation instance. The simulation is starded by running the function coresponding to the simulation time: RunOneRouteMultipleTimes or RunMultipleRoutes.

After the simulation is started the routeSerializerThread and updateSerializeThread threads are started. These two threads will be running for the entire length of the simulation. They waiting for requests from Traffic Participants and serving them in the order they are added to the lists.

The simulation is composed of multiple smaller simulations, one for every threshold from the start threshold simulation parameter to the end threshold, with an increment of 10. This means the number traffic participants taking the „best” route over the „shortest” one increases from iteration to iteration.

For each iteration Traffic Participants are generated. Each one is given an id, a start delay, a start location, an end location and a behavior. The behavior is a random between greedy and not greedy with a chance of threshold percent of being not greedy. After all the Traffic Participants are instantiated, the RunTrafficParticipant method of each one of them is run on a separate thread.

Each Traffic participant sleeps for the start delay it was assigned. This sleep is denoted as a gray area on the lifeline of the Traffic Participant in Figure 5-9. Then the Traffic Participant requests their first route, from the start location to the end location. This is done by calling the AddRouteRequest method in the RequestSerializer and waiting to receive an answer. The request is added to the routeRequests list and processed by the routeSerializerThread. This thread takes every request in the routeRequests list, converts it into a HTTP call and sends it to the GetRoute endpoint of the Live Traffic Server. After a route is received it is deserialized from the GeoJson into a RouteModel instance. That instance is assigned to the traffic participant that issued the request. The request is removed from the waiting list.

After a Traffic Participant receives a route it starts following it, going feature by feature in the route. The Traffic Participant waits for the time it takes for it to travel the length of the feature. This wait period is also marked in gray on the Traffic Participants lifeline in the sequence diagram. Then sends an update request that it left one feature and it is joining another. These features are represented by the coordinates of their middle. The first update is done by sending (0,0) as the coordinate for the previous edge. The last update is done by sending (0,0) as the coordinate for the current edge. This can be seen in Figure

5-9 colored in red. The update requests are done by calling the AddUpdateRequest method in the RequestSerializer. Each update request is processed by the updateSerializerThread similar to the way route requests are handled by the routeSerializerThread.

After all the Traffic Participants have finished their routes, the method in the Simulation class continues its flow. It collects the statistics from every Traffic Participant and from the RequestSerializer and computes the general statistics for the current threshold. This repeats for every other threshold.

Chapter 6. Testing and Validation

6.1. Hardware

Even though TCS-3S is a distributable system, the testing and validation phase of the development was done on a single machine at a time. Three machines with different configurations were used in the development and testing of the system.

1. Intel Core i5-4460 3.20 GHz, 16 GB RAM, 100 GB free SSD storage
2. Intel Core i7-4720 2.60 GHz, 16 GB RAM, 60 GB free SSD storage
3. Intel Core i5-6700 2.60 GHz, 16 GB RAM, 100 GB free SSD storage

6.2. Metrics

In order to study the efficiency of the presented project, metrics should be defined for both the system itself and the effects of the system in the connected cars context, computed using the implemented Traffic Simulator (4.3.1.).

6.2.1. System metrics

The system metrics are the characteristics of the system, measuring the way this works. These metrics would show the capabilities of the system in any module configuration. They focus especially on the Live Traffic Server and on the Route Computing Server.

Metric Name	Description
Number of Route Requests	The number of route requests received by the Live Traffic Server over the length of the simulation
Average Route Request Response Time	The average time it takes for the Live Traffic Server to respond to a request for a route.
Maximum Route Request Response Time	The largest amount of time it took the Live Traffic Server to respond to a Route Request
Number of Update Requests	The number of update requests received by the Live Traffic Server over the length of the simulation
Average Update Request Response Time	The average time it takes for the Live Traffic Server to respond to an update for a route.
Maximum Update Request Response Time	The largest amount of time it took the Live Traffic Server to respond to a Route Request

Table 6-1 Definitions of System Metrics in TCS-3C

6.2.2. Simulation metrics

The Simulation metrics are the characteristics of the results obtained by running different scenarios in the Traffic Simulator. These metrics are used to analyse and compare different approaches of implementation or different traffic scenarios.

Metric Name	Description
Threshold (T)	The percentage of people picking the “best” route over the “shortest” route
Simulation Length (SL)	The time it takes to simulate the scenario with the given simulation parameters.
Total Time (TT)	The sum of the durations of all routes simulated.
Average Route Time (ART)	The average duration of the routes simulated
Average Speed (AS)	The average speed of the Traffic Participants simulated
Average Touched Features (ATF)	The average number of edge sets touched by the Traffic participants while following the simulated routes

Table 6-2 Definitions of Simulation Metrics in TCS-3C

6.3. Scenarios

The testing and validation phase of the development includes two different scenarios. These scenarios aim to create a context that facilitates the study of the results of the system, based on the metrics described at point 6.2.1. Furthermore, the contexts created in the scenarios give the chance to study the effects of the choices made by traffic participants in the Connected Cars Context.

- **The Run One Route Multiple Times scenario** describes a context in which multiple cars request the exact same route with a small delay.

  The start location of the scenario is on street Alexandru Vaida Voevod in Cluj-Napoca, Romania. The end location of the route is in Cipariu square in Cluj-Napoca Romania

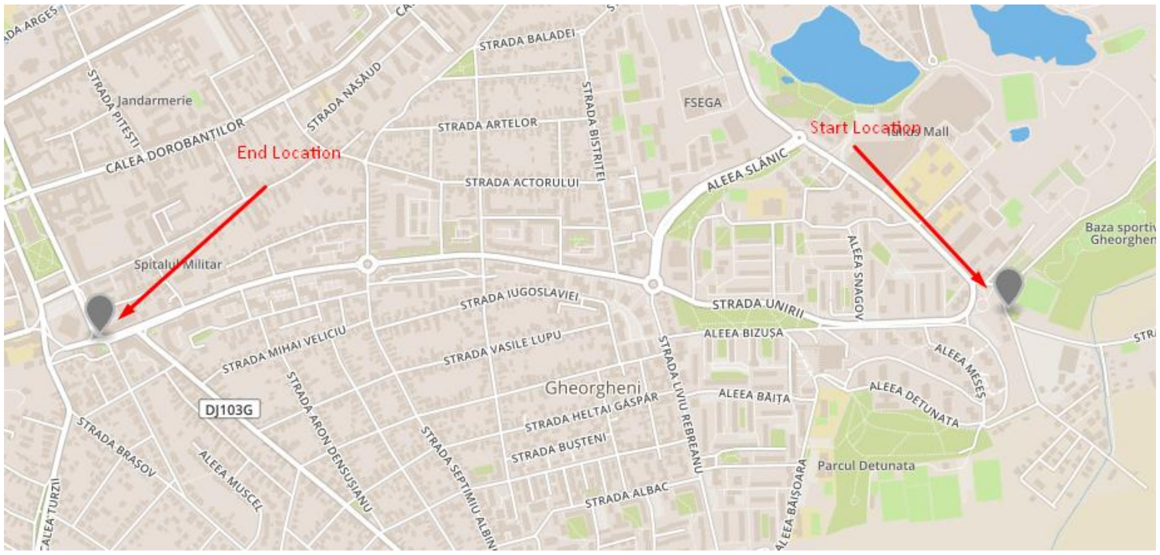


Figure 6-1 Screenshot of the Map Contining Start and End Location of First Scenario

The simulation parameters for this scenario are described in Table 6-1.

Number of cars	100
Request delay	00:00:15
Simulation type	OneRouteMultipleTimes
Time multiplyer	1
Stat threshold	0
End threshold	100

Table 6-3 Simulation Parameters of First Scenario

- **The Run Multiple Routes scenario** describes a context in which multiple cars take multiple routes between four locations on the map. The chosen start and end location are further away from one another than the ones used in the first scenario. The traffic participants are randomly assigned a start location from the two available ones and they are randomly assigned an end location from the two available ones. These locations can be seen in Figure 6-2.

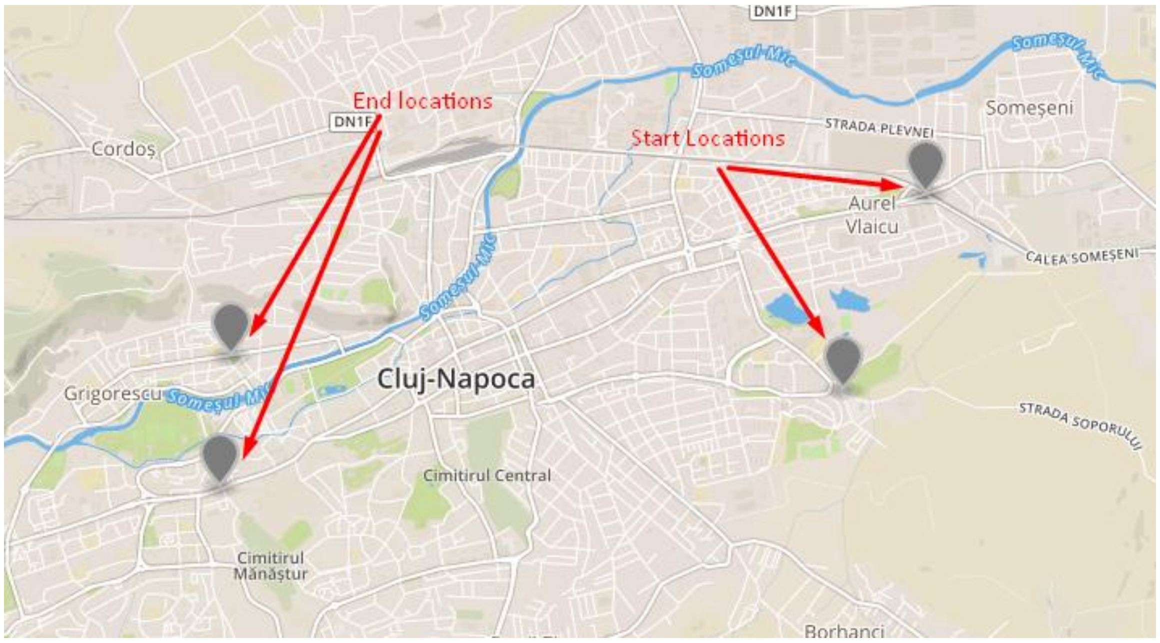


Figure 6-2 Screenshot of the Map Containing the Start and End Locations in the Second Scenario

The simulation parameters used in the Run Multiple Routes scenario are presented in Table 6-2.

Number of cars	500
Request delay	00:00:10
Simulation type	MultipleRoutes
Time multiplyer	1
Stat threshold	0
End threshold	100

Table 6-4 Simulation Parameters of Second Scenario

6.4. Results

By running the system on a single machine and applying the aforementioned scenarios, results have been obtained for both the system and the simulation. These results are received with respect to the Metrics defined at Point 6.2.

- **Run One Route Multiple Times scenario**
  - The System Results obtained by the simulation of the Run One Route Multiple times scenario are presented in Table 6-5.

Number of Route Requests	15072
Average Route Request Response Time	00:00:00.120293
Maximum Route Request Response Time	00:00:02.274660
Number of Update Requests	52231
Average Update Request Response Time	00:00:00.0192022
Maximum Update Request Response Time	00:00:00.125830

Table 6-5 Sistem Results of the First Scenario

  - The Simulation results obtained by the simulator when running the Run One Route Multiple Times scenario are presented in Table 6-6

T	TT	ART	AS	ATF
0	05:08:15.817	00:03:04.158	49.83	49
10	05:08:50.452	00:03:04.492	49.76	49
20	05:08:34.218	00:03:05.142	49.84	49
30	05:08:52.413	00:03:05.324	49.84	49
40	05:08:11.404	00:03:04.914	49.85	49
50	05:07:49.493	00:03:04.694	49.91	49
60	05:09:05.774	00:03:05.457	49.83	49
70	05:10:06.993	00:03:06.069	49.71	49
80	05:08:06.417	00:03:04.864	49.88	49
90	05:08:08.504	00:03:04.885	49.84	49
100	05:08:10.976	00:03:04.909	49.84	49

Table 6-6 Simulation Results of the First Scenario

- **Run Multiple Routes scenario**
  - The System Results obtained by the simulation of the Run Multiple Routes scenario are presented in Table 6-7.

Number of Route Requests	183722
Average Route Request Response Time	00:00:00.112195
Maximum Route Request Response Time	00:00:06.66811
Number of Update Requests	573650
Average Update Request Response Time	00:00:00.0157105
Maximum Update Request Response Time	00:00:00.317340

Table 6-7 Sistem Results of the Second Scenario

- The Simulation results obtained by the simulator when running the Run One Route Multiple Times scenario are presented in Table 6-8

T	TT	ART	AS	ATF
0	2.14:31:04.817	00:07:29.246	49.53	89.104
10	2.14:24:55.216	00:07:28.366	49.50	89.246
20	2.14:57:14.687	00:07:25.942	48.69	89.246
30	2.14:30:47.172	00:07:28.982	49.55	88.986
40	2.14:21:28.050	00:07:28.177	49.41	89.984
50	2.14:16:15.306	00:07:27.296	49.53	89.832
60	2.14:41:21.047	00:07:30.221	49.40	89.392
70	2.12:43:53.417	00:07:31.176	49.46	89.452
80	2.14:38:12.258	00:07:30.258	49.49	89.236
90	2.14:35:27.213	00:07:29.502	49.72	89.454
100	2.14:32:12.258	00:07:29.365	49.84	89.246

Table 6-8 Simulation Results of the Second Scenario

The results obtained by the system are analyzed using the metrics defined, for both the system and the simulation. These results are used for further improving the system, the approach on traffic data interpretation and the proposed scenarios.

The results obtained for the system are satisfying with respect to the Nonfunctional Requirements proposed at Point 2.2.2. The average waiting time is under 0.2 seconds for both the presented scenarios. The system processes around 4 requests every second, but the fact that the route and update requests are serialized in the simulator needs to be taken into consideration.

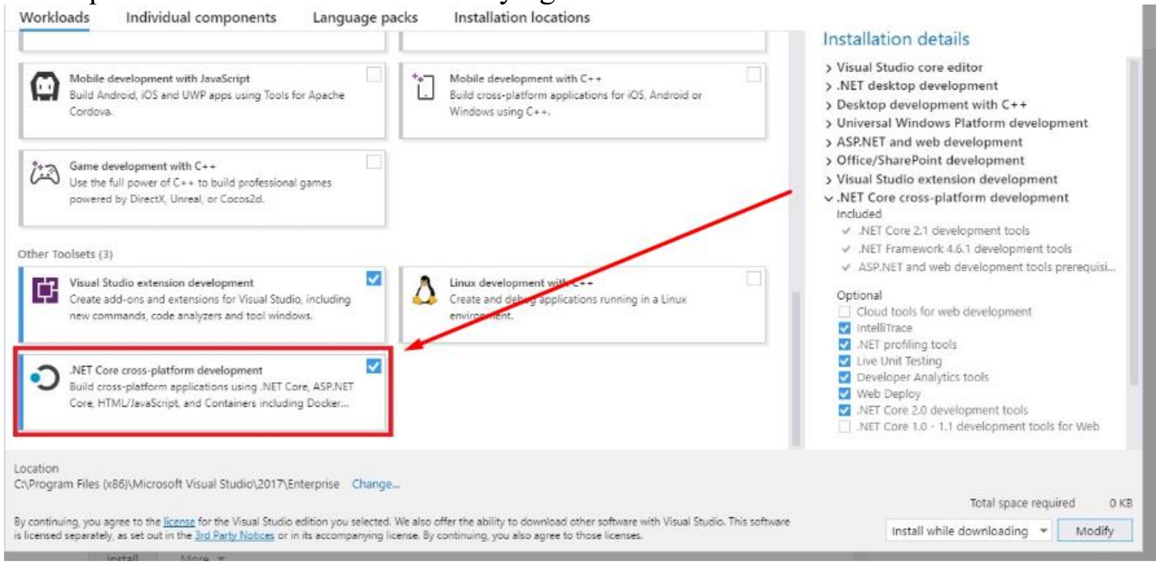
On the other hand, the results obtained for the simulation show that this approach on the study of behaviors of traffic participants in the Connected Cars Context is not a valid one. This can be observed from the fact that the results obtained for the Average Route Time metric does not linearly decrease with the increase of the threshold. The source of this problem can come for different points. First of all, the simulation scenarios in combination with the custom_speed function defined at Point 4.3.2 fail to simulate traffic congestions that really affect the Connected Cars Context. Another explanation of the obtained results can be the fact that no estimations of the future traffic status are made. The routing is done by considering that an edge will be exactly as crowded when the traffic participant will reach it as it is in the moment of the request. The repetitive route requests try to minimize this problem. Even so, the traffic participant might take a longer route in order to avoid crowded segments that eventually discharge and the traffic participant ends up spending more time than they should on the proposed route. This Subject needs to be further investigated.

Chapter 7. User's manual

This chapter comprises the installation of all the components of the developed system as well as methods to test each one of these components. Furthermore, all the development and testing were done on only one machine at a time, with some supplementary steps the system can be easily distributed. The choice of testing only on one machine at a time was made so the speed of the simulation was as big as possible.

7.1. Environment

For running of the current version of the system the user needs the Visual Studio IDE. The development was done using Microsoft Visual Studio Express 2017. Using a different version of Visual Studio might create issues. .NET Core 2.0 was used in the development of the Live Traffic Server and Route Computing server so the user must make sure the .NET Core SDK is installed, by checking the ".NET Core cross-platform development" in the installation/modifying wizard of Visual Studio.



Mobile development with JavaScript
Build Android, iOS and UWP apps using Tools for Apache Cordova.

Mobile development with C++
Build cross-platform applications for iOS, Android or Windows using C++.

Game development with C++
Use the full power of C++ to build professional games powered by DirectX, Unreal, or Cocos2d.

Other Toolsets (3)

[✓] Visual Studio extension development
Create add-ons and extensions for Visual Studio, including new commands, code analyzers and tool windows.

Linux development with C++
Create and debug applications running in a Linux environment.

[✓] .NET Core cross-platform development
Build cross-platform applications using .NET Core, ASP.NET Core, HTML/JavaScript, and Containers including Docker...

Installation details
- Visual Studio core editor
- .NET desktop development
- Desktop development with C++
- Universal Windows Platform development
- ASP.NET and web development
- Office/SharePoint development
- Visual Studio extension development
- .NET Core cross-platform development
  - Included
    - ✓ .NET Core 2.1 development tools
    - ✓ .NET Framework 4.6.1 development tools
    - ✓ ASP.NET and web development tools prerequisi...
  - Optional
    - [ ] Cloud tools for web development
    - [✓] IntelliTrace
    - [✓] .NET profiling tools
    - [✓] Live Unit Testing
    - [✓] Developer Analytics tools
    - [✓] Web Deploy
    - [✓] .NET Core 2.0 development tools
    - [ ] .NET Core 1.0 - 1.1 development tools for Web

Location
C:\Program Files (x86)\Microsoft Visual Studio\2017\Enterprise

By continuing, you agree to the license for the Visual Studio edition you selected. We also offer the ability to download other software with Visual Studio. This software is licensed separately, as set out in the 3rd Party Notices or in its accompanying license. By continuing, you also agree to those licenses.

Total space required 0 KB

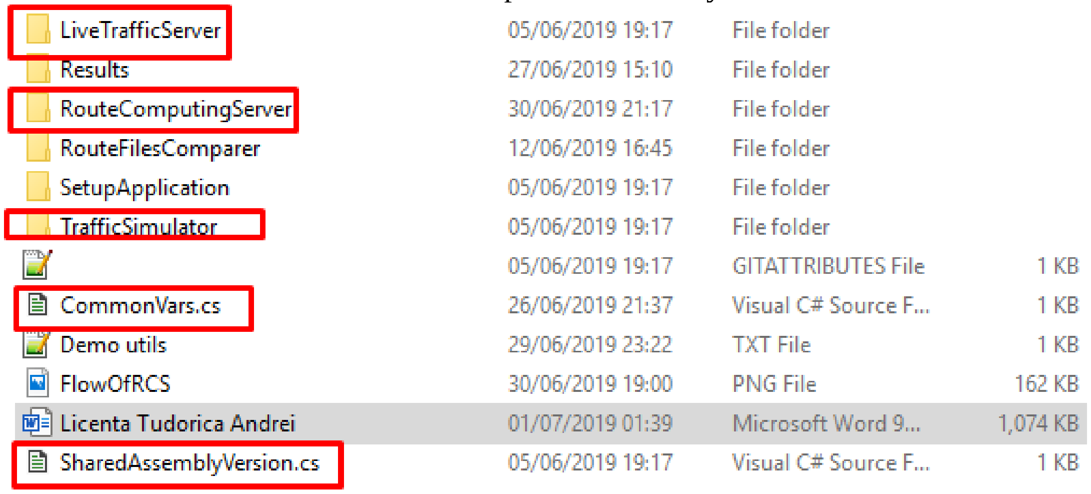
Figure 7-1 Visual Studio installation wizzard

7.2. Dependencies

Each module of the system uses a few external elements. These elements have to be present in order for the system to work. In the case of a distribution of the system over multiple machines the following steps must be done for each machine.

- Download the project from [24]
- Create a folder destinated for map related files
- Create a folder destinated for the results of the simulation
- Download the map of the country of focus from [25]
- Reduce the map to the specific area of focus using instructions from [26] (the smaller the area, the faster the system)
- Add the .pbf file of the area to the folder created at step 1.

- Make sure that the CommonVars.cs and SharedAssemblyVersion.cs files are in the same folder with the components of the system on the machine.



LiveTrafficServer	05/06/2019 19:17	File folder	
Results	27/06/2019 15:10	File folder	
RouteComputingServer	30/06/2019 21:17	File folder	
RouteFilesComparer	12/06/2019 16:45	File folder	
SetupApplication	05/06/2019 19:17	File folder	
TrafficSimulator	05/06/2019 19:17	File folder	
	05/06/2019 19:17	GITATTRIBUTES File	1 KB
CommonVars.cs	26/06/2019 21:37	Visual C# Source F...	1 KB
Demo utils	29/06/2019 23:22	TXT File	1 KB
FlowOfRCS	30/06/2019 19:00	PNG File	162 KB
Licenta Tudorica Andrei	01/07/2019 01:39	Microsoft Word 9...	1,074 KB
SharedAssemblyVersion.cs	05/06/2019 19:17	Visual C# Source F...	1 KB

Figure 7-2 Screenshot of the folder containing components and common files

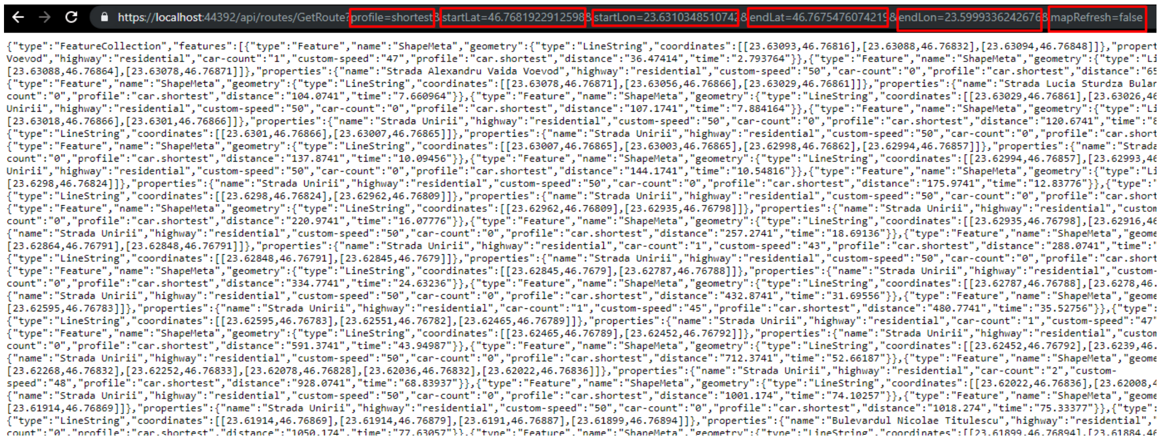
- Open the CommonVars.cs file
- Change the value of “PathToCommonFolder” to be the path of the folder you created for the maps
- Change the value of “PathToResultsFolder” to be the path of the folder you created for the results
- Add the custom-car.lua file, that can be found in the root folder of the system, in the folder destinated for maps
- Change the “PbfFileName” value to be equal to the name of the final map to be used
- Change the “RouterDbFileName” to match with the target area
- Run the SetupApplication found in the same folder with all the components

Name	**Value**
PathToCommonFolder	“D:\\Andrei\\Scoala\\LICENTA\\Maps\\”
PathToResultsFolder	“D:\\Andrei\\Scoala\\LICENTA\\TCCC\\Results\\”
CustomCarProfileFileName	“custom-car.lua”
PbfMapFileName	“cluj-napoca.pbf”
RouterDbFileName	“cluj-napoca.routerdb”
UpdatesNecesaryForMapRefresh	50

Table 7-1 Fields in CommonVars.cs in current version

7.3. Route Computing Server

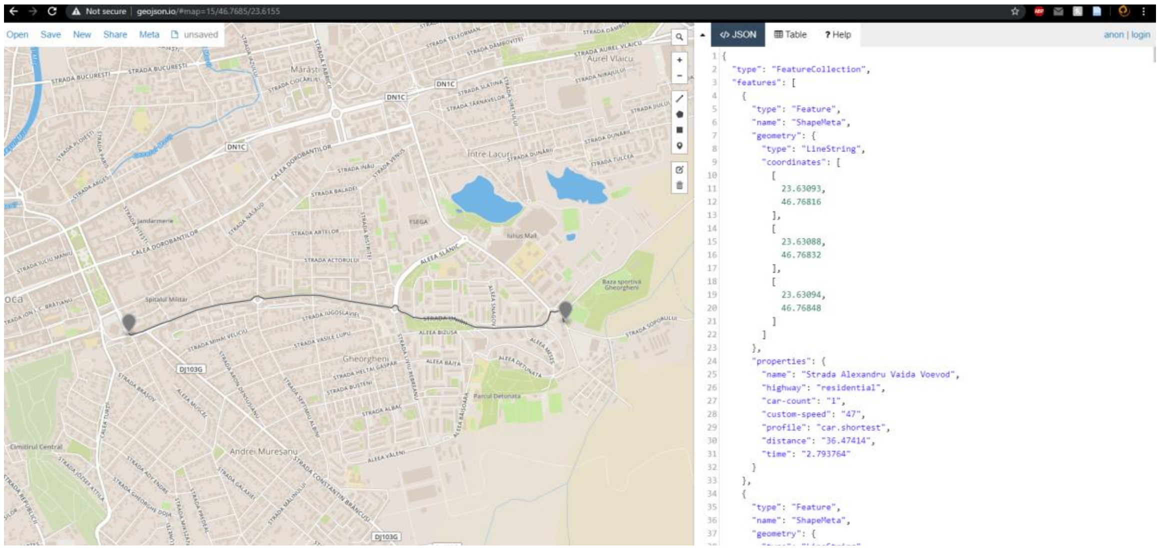
The Route Computing Server code can be found in the "RouteComputingServer" folder of the downloaded system. After all the steps at 7.2. are completed, run the .sln file and compile the project. This should open a page saying the Route Computing Server is running and waiting for commands. Testing can be done by building a link containing 2 locations on the chosen map.



```
https://localhost:44392/api/routes/GetRoute?profile=shortest&startLat=46.7681922912598&startLon=23.6310348510742&endLat=46.7675476074219&endLon=23.5999336242676&mapRefresh=false

{"type":"FeatureCollection","features":[{"type":"Feature","name":"ShapeMeta","geometry":{"type":"LineString","coordinates":[[23.63093,46.76816],[23.63088,46.76832],[23.63094,46.76848]]},"propert
Voevod","highway":"residential","car-count":"1","custom-speed":"47","profile":"car.shortest","distance":"36.47414","time":"2.793764"}},{"type":"Feature","name":"ShapeMeta","geometry":{"type":"Li
[23.63088,46.76864],[23.63078,46.76871]]},"properties":{"name":"Strada Alexandru Vaida Voevod","highway":"residential","custom-speed":"50","car-count":"0","profile":"car.shortest","distance":"65
{"type":"Feature","name":"ShapeMeta","geometry":{"type":"LineString","coordinates":[[23.63078,46.76871],[23.63056,46.76866],[23.63029,46.76861]]},"properties":{"name":"Strada Lucia Sturdza Bular
count":"0","profile":"car.shortest","distance":"104.0741","time":"7.660964"}},{"type":"Feature","name":"ShapeMeta","geometry":{"type":"LineString","coordinates":[[23.63029,46.76861],[23.63026,46
Unirii","highway":"residential","custom-speed":"50","car-count":"0","profile":"car.shortest","distance":"107.1741","time":"7.884164"}},{"type":"Feature","name":"ShapeMeta","geometry":{"type":"Li
[23.63018,46.76866],[23.6301,46.76866]]},"properties":{"name":"Strada Unirii","highway":"residential","custom-speed":"50","car-count":"0","profile":"car.shortest","distance":"120.6741","time":"8
{"type":"LineString","coordinates":[[23.6301,46.76866],[23.63007,46.76865]]},"properties":{"name":"Strada Unirii","highway":"residential","custom-speed":"50","car-count":"0","profile":"car.short
{"type":"Feature","name":"ShapeMeta","geometry":{"type":"LineString","coordinates":[[23.63007,46.76865],[23.63003,46.76865],[23.62998,46.76862],[23.62994,46.76857]]},"properties":{"name":"Strada
count":"0","profile":"car.shortest","distance":"137.8741","time":"10.09456"}},{"type":"Feature","name":"ShapeMeta","geometry":{"type":"LineString","coordinates":[[23.62994,46.76857],[23.62993,46
Unirii","highway":"residential","custom-speed":"50","car-count":"0","profile":"car.shortest","distance":"144.1741","time":"10.54816"}},{"type":"Feature","name":"ShapeMeta","geometry":{"type":"Li
[23.6298,46.76824]]},"properties":{"name":"Strada Unirii","highway":"residential","custom-speed":"50","car-count":"0","profile":"car.shortest","distance":"175.9741","time":"12.83776"}},{"type":"
{"type":"LineString","coordinates":[[23.6298,46.76824],[23.62962,46.76809]]},"properties":{"name":"Strada Unirii","highway":"residential","custom-speed":"50","car-count":"0","profile":"car.short
{"type":"Feature","name":"ShapeMeta","geometry":{"type":"LineString","coordinates":[[23.62962,46.76809],[23.62935,46.76798]]},"properties":{"name":"Strada Unirii","highway":"residential","custom
count":"0","profile":"car.shortest","distance":"220.9741","time":"16.07776"}},{"type":"Feature","name":"ShapeMeta","geometry":{"type":"LineString","coordinates":[[23.62935,46.76798],[23.62916,46
{"name":"Strada Unirii","highway":"residential","custom-speed":"50","car-count":"0","profile":"car.shortest","distance":"257.2741","time":"18.69136"}},{"type":"Feature","name":"ShapeMeta","geome
[23.62864,46.76791],[23.62848,46.76791]]},"properties":{"name":"Strada Unirii","highway":"residential","car-count":"1","custom-speed":"43","profile":"car.shortest","distance":"288.0741","time":"
{"type":"LineString","coordinates":[[23.62848,46.76791],[23.62845,46.7679]]},"properties":{"name":"Strada Unirii","highway":"residential","custom-speed":"50","car-count":"0","profile":"car.short
{"type":"Feature","name":"ShapeMeta","geometry":{"type":"LineString","coordinates":[[23.62845,46.7679],[23.62787,46.76788]]},"properties":{"name":"Strada Unirii","highway":"residential","custom-
count":"0","profile":"car.shortest","distance":"334.7741","time":"24.63236"}},{"type":"Feature","name":"ShapeMeta","geometry":{"type":"LineString","coordinates":[[23.62787,46.76788],[23.6278,46.
{"name":"Strada Unirii","highway":"residential","custom-speed":"50","car-count":"0","profile":"car.shortest","distance":"432.8741","time":"31.69556"}},{"type":"Feature","name":"ShapeMeta","geome
[23.62595,46.76783]]},"properties":{"name":"Strada Unirii","highway":"residential","car-count":"1","custom-speed":"45","profile":"car.shortest","distance":"480.7741","time":"35.52756"}},{"type":
{"type":"LineString","coordinates":[[23.62595,46.76783],[23.62551,46.76782],[23.62465,46.76789]]},"properties":{"name":"Strada Unirii","highway":"residential","car-count":"1","custom-speed":"47"
{"type":"Feature","name":"ShapeMeta","geometry":{"type":"LineString","coordinates":[[23.62465,46.76789],[23.62452,46.76792]]},"properties":{"name":"Strada Unirii","highway":"residential","custom
count":"0","profile":"car.shortest","distance":"591.3741","time":"43.94987"}},{"type":"Feature","name":"ShapeMeta","geometry":{"type":"LineString","coordinates":[[23.62452,46.76792],[23.6239,46.
{"name":"Strada Unirii","highway":"residential","custom-speed":"50","car-count":"0","profile":"car.shortest","distance":"712.3741","time":"52.66187"}},{"type":"Feature","name":"ShapeMeta","geome
[23.62268,46.76832],[23.62252,46.76833],[23.62078,46.76828],[23.62036,46.76832],[23.62022,46.76836]]},"properties":{"name":"Strada Unirii","highway":"residential","car-count":"2","custom-
speed":"48","profile":"car.shortest","distance":"928.0741","time":"68.83937"}},{"type":"Feature","name":"ShapeMeta","geometry":{"type":"LineString","coordinates":[[23.62022,46.76836],[23.62008,4
{"name":"Strada Unirii","highway":"residential","custom-speed":"50","car-count":"0","profile":"car.shortest","distance":"1001.174","time":"74.10257"}},{"type":"Feature","name":"ShapeMeta","geome
[23.61914,46.76869]]},"properties":{"name":"Strada Unirii","highway":"residential","custom-speed":"50","car-count":"0","profile":"car.shortest","distance":"1018.274","time":"75.33377"}},{"type":
{"type":"LineString","coordinates":[[23.61914,46.76869],[23.61914,46.76879],[23.6191,46.76887],[23.61899,46.76894]]},"properties":{"name":"Bulevardul Nicolae Titulescu","highway":"residential","
count":"0","profile":"car.shortest","distance":"1050.174","time":"77.63057"}},{"type":"Feature","name":"ShapeMeta","geometry":{"type":"LineString","coordinates":[[23.61899,46.76894],[23.61884,46
```

Figure 7-3 Screenshot of test run for the Route Computing Server

Figure 7.4. presents an example of usage for the following link: https://localhost:44392/api/routes/GetRoute?profile=shortest&startLat=46.7681922912598&startLon=23.6310348510742&endLat=46.7675476074219&endLon=23.5999336242676&mapRefresh=false. The response of the Route Computing Server is a GeoJson string containing a route. By pasting this into [27] the visualization of this route is obtained.



```
{
  "type": "FeatureCollection",
  "features": [
    {
      "type": "Feature",
      "name": "ShapeMeta",
      "geometry": {
        "type": "LineString",
        "coordinates": [
          [
            23.63093,
            46.76816
          ],
          [
            23.63088,
            46.76832
          ],
          [
            23.63094,
            46.76848
          ]
        ]
      },
      "properties": {
        "name": "Strada Alexandru Vaida Voevod",
        "highway": "residential",
        "car-count": "1",
        "custom-speed": "47",
        "profile": "car.shortest",
        "distance": "36.47414",
        "time": "2.793764"
      }
    },
    {
      "type": "Feature",
      "name": "ShapeMeta",
      "geometry": {
```

Figure 7-4 Screenshot of the visualization of the response received by the command in Figure 7.4.

For best performance the Route Computing server should be done in Release mode. This can be done by selecting „Release" in the run mode dropdown menu and pressing CTRL+F5.

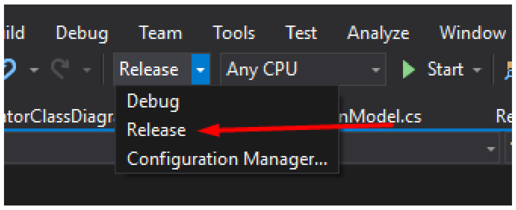


Figure 7-5 screenshot of run mode dropdown menu in Visula Studio

7.4. Live Traffic Server

The Live Traffic Server code can be found in the “LiveTrafficServer” folder of the downloaded system. After all the steps at 7.2. are completed, open the .sln file. Open the Constants.cs file in Visual Studio and modify the value of “RouteComputingServerURL” so that it matches the URL where the Route Computing Server. In a local environment this is "https://localhost:44392/".

When the project is run a page saying “Live Traffic Server is Running. Waiting for commands.” should open.

The server initializes the maps by itself at start up. The functionality can be tested by building a link containing 2 locations on the chosen map just like the one for the Route Computing server.

Example:
https://localhost:44351/api/traffic/GetRoute?profile=shortest&startLat=46.7681922912598&startLon=23.63103485107422&endLat=46.7675476074219&endLon=23.5999336242676

The visualization of the answer follows the same steps as the ones for the Route Computing Server.

For best performance the Route Computing server should be run in Release mode. This can be done by selecting „Release” in the run mode dropdown menu (Figure 7.6) and pressing CTRL+F5.

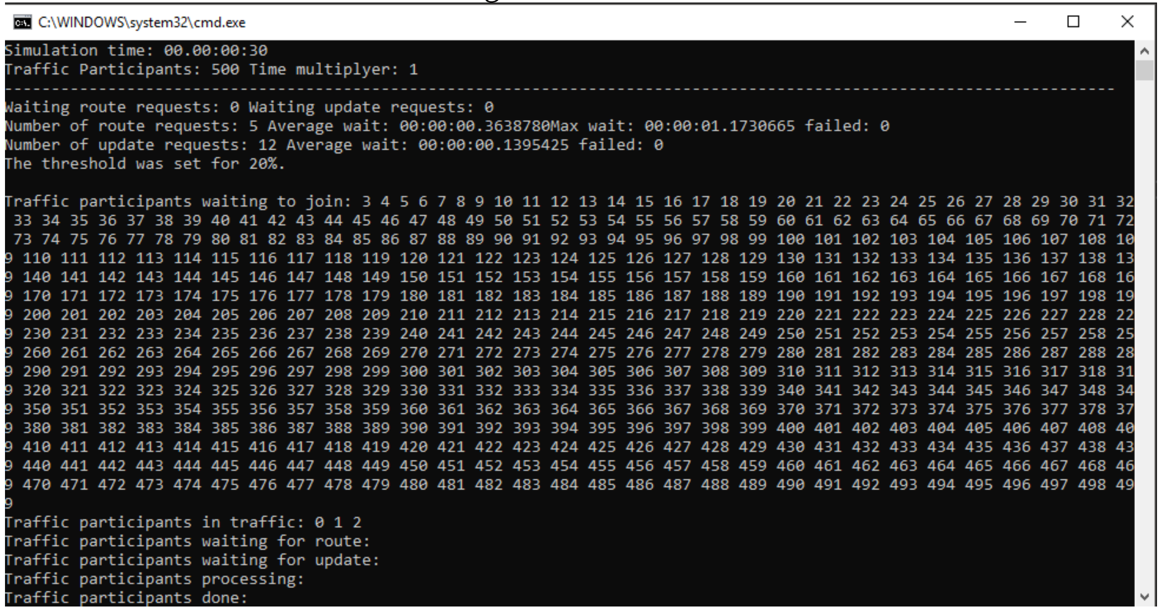
7.5. Traffic Simulation

The Traffic Simulator code can be found in the “TrafficSimulator” folder. After all the steps at 7.2. are completed, open the .sln file. Open the SimulationParameters.json file in Visual Studio and modify the value of the “LiveTrafficServerUri” such that it matches the URL where the running Live Traffic Server can be found. In a local environment that link should be "https://localhost:44351". Modify the other simulation parameters so they match the desired simulation.

NumberOfCars	500
RequestDelay	00:00:10
LiveTrafficServerUri	https://localhost:44351
SimulationType	2
timeMultiplyer	1
startTH	20
endTH	100

Table 7-1 Example of simulations parameters

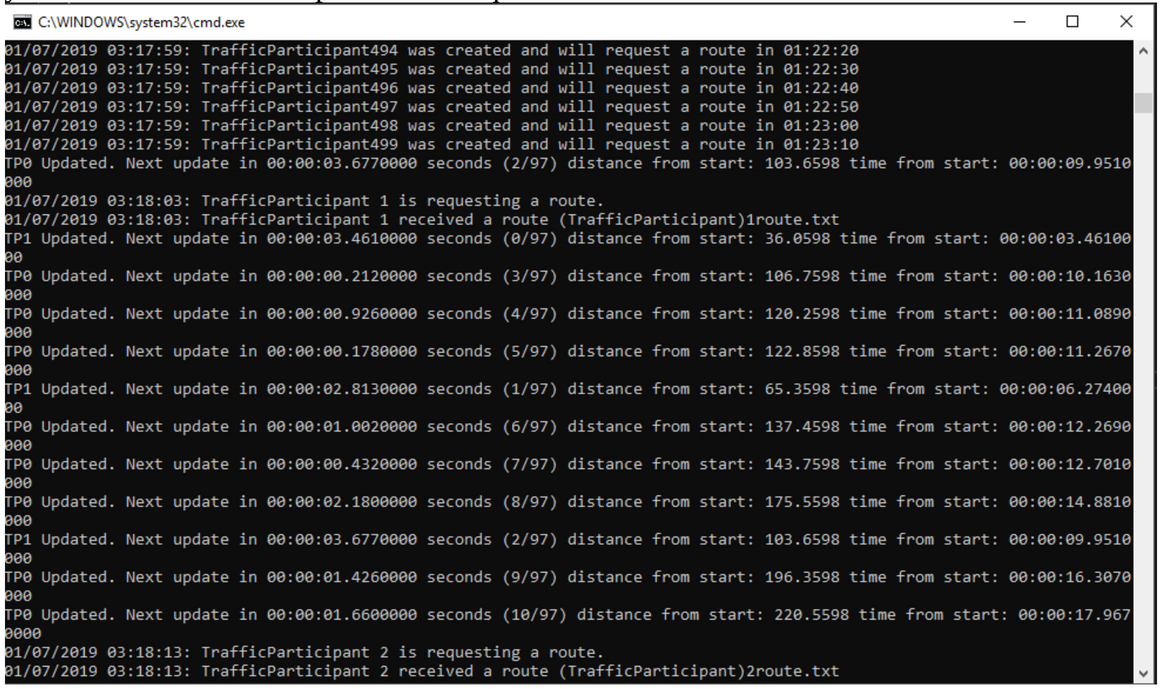
Run the simulator in release mode to get live statistics about the simulation.



```
C:\WINDOWS\system32\cmd.exe
Simulation time: 00.00:00:30
Traffic Participants: 500 Time multiplyer: 1
-----------------------------------------------------------------------------------------------------------------------
Waiting route requests: 0 Waiting update requests: 0
Number of route requests: 5 Average wait: 00:00:00.3638780Max wait: 00:00:01.1730665 failed: 0
Number of update requests: 12 Average wait: 00:00:00.1395425 failed: 0
The threshold was set for 20%.

Traffic participants waiting to join: 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32
 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72
 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100 101 102 103 104 105 106 107 108 10
9 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 13
9 140 141 142 143 144 145 146 147 148 149 150 151 152 153 154 155 156 157 158 159 160 161 162 163 164 165 166 167 168 16
9 170 171 172 173 174 175 176 177 178 179 180 181 182 183 184 185 186 187 188 189 190 191 192 193 194 195 196 197 198 19
9 200 201 202 203 204 205 206 207 208 209 210 211 212 213 214 215 216 217 218 219 220 221 222 223 224 225 226 227 228 22
9 230 231 232 233 234 235 236 237 238 239 240 241 242 243 244 245 246 247 248 249 250 251 252 253 254 255 256 257 258 25
9 260 261 262 263 264 265 266 267 268 269 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 28
9 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 31
9 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 34
9 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 37
9 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 40
9 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 43
9 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 46
9 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 49
9

Traffic participants in traffic: 0 1 2
Traffic participants waiting for route:
Traffic participants waiting for update:
Traffic participants processing:
Traffic participants done:
```

Figure 7-6 Screenshot of simulator run in Release mode

Run the simulator in Debug mode to follow the Simulation step by step. This might be really hard to follow as the updates move quite fast.



```
C:\WINDOWS\system32\cmd.exe
01/07/2019 03:17:59: TrafficParticipant494 was created and will request a route in 01:22:20
01/07/2019 03:17:59: TrafficParticipant495 was created and will request a route in 01:22:30
01/07/2019 03:17:59: TrafficParticipant496 was created and will request a route in 01:22:40
01/07/2019 03:17:59: TrafficParticipant497 was created and will request a route in 01:22:50
01/07/2019 03:17:59: TrafficParticipant498 was created and will request a route in 01:23:00
01/07/2019 03:17:59: TrafficParticipant499 was created and will request a route in 01:23:10
TP0 Updated. Next update in 00:00:03.6770000 seconds (2/97) distance from start: 103.6598 time from start: 00:00:09.9510
000
01/07/2019 03:18:03: TrafficParticipant 1 is requesting a route.
01/07/2019 03:18:03: TrafficParticipant 1 received a route (TrafficParticipant)1route.txt
TP1 Updated. Next update in 00:00:03.4610000 seconds (0/97) distance from start: 36.0598 time from start: 00:00:03.46100
00
TP0 Updated. Next update in 00:00:00.2120000 seconds (3/97) distance from start: 106.7598 time from start: 00:00:10.1630
000
TP0 Updated. Next update in 00:00:00.9260000 seconds (4/97) distance from start: 120.2598 time from start: 00:00:11.0890
000
TP0 Updated. Next update in 00:00:00.1780000 seconds (5/97) distance from start: 122.8598 time from start: 00:00:11.2670
000
TP1 Updated. Next update in 00:00:02.8130000 seconds (1/97) distance from start: 65.3598 time from start: 00:00:06.27400
00
TP0 Updated. Next update in 00:00:01.0020000 seconds (6/97) distance from start: 137.4598 time from start: 00:00:12.2690
000
TP0 Updated. Next update in 00:00:00.4320000 seconds (7/97) distance from start: 143.7598 time from start: 00:00:12.7010
000
TP0 Updated. Next update in 00:00:02.1800000 seconds (8/97) distance from start: 175.5598 time from start: 00:00:14.8810
000
TP1 Updated. Next update in 00:00:03.6770000 seconds (2/97) distance from start: 103.6598 time from start: 00:00:09.9510
000
TP0 Updated. Next update in 00:00:01.4260000 seconds (9/97) distance from start: 196.3598 time from start: 00:00:16.3070
000
TP0 Updated. Next update in 00:00:01.6600000 seconds (10/97) distance from start: 220.5598 time from start: 00:00:17.967
0000
01/07/2019 03:18:13: TrafficParticipant 2 is requesting a route.
01/07/2019 03:18:13: TrafficParticipant 2 received a route (TrafficParticipant)2route.txt
```

Figure 7-7 screenshot of simulator run in Debug mode

The results of the simulation can be found on the console during the simulation and at the end of it. Also, at the end of the simulation, in the Results folder that was created at 7.2, the simulator creates files for the entire simulation statistics and for each traffic participant.

For best performance the Simulator should be run in Release mode.

7.6. Distribution

In order to distribute the project over multiple machines, the following steps have to be followed:

- Follow the installation guidance for the Live Traffic Server and for the Route Computing Server on the designated machines.
- Make port forwarding for the ports of the distributed servers. Steps can be found in [28].
- Add UseUrls feature to the CreateDefaultBuilder method in the Program.cs class.
  These steps can also be found in [29].

Chapter 8. Conclusions and Future Work

The topic of the Connected Cars Context and Intelligent Transportation Systems (ITS) is a highly researched one. The resource collection for these domains is rich in research papers concerning different subtopics such as data analysis, route computing techniques, connected car systems, infrastructure elements, etc. The existing software solutions, on the other hand, fail to freely offer the possibility of centralized research and development.

The proposed solution addresses the previously mentioned problem by the system described in this document. TCS-3C presents the core of a distributed system open to accommodate various modules that add new features for the best interest of Traffic Researchers, Traffic Participants and Traffic Supervisors.

The scope of this project was to design, develop and test a system that covers aids the research activities in the Connected Cars Context, by building over the already existing Solution of Itinero. The project also aimed to study the effect of the route choosing behaviors of the traffic participants in the Connected Cars Context.

The contributions brought to this project will be studied with respect to the objectives described at 2.1. These contributions will be marked in table 8-1 with a green tick (✓) if the objectives were done, a red X if the objectives were not completed. These marks are related to the associated type of challenges. A black line signals there was no challenge to be tackled.

OBJECTIVE	CS	CD
O1.1	✓	-
O1.2	✓	✓
O1.3	-	✓
O1.4	-	✓
O1.5	-	✓
O1.6	✓	-
O1.7	-	✓
O1.8	-	✓
O2.1	✓	-
O2.2	✓	✓
O2.3	-	✓
O3.1	✓	-
O3.2	✓	-
O3.3	✓	-
O3.4	-	✓
O3.5	✓	-
O3.6	X	-

Table 8-1 Objectives, Challenges and Contributions

The only challenge that raised problems was the O3.6. Due to the fact that the data obtained at O3.5 did not resemble the required information, no relevant conclusion could be drawn concerning the effects of traffic participants choices in the Connected Cars Context, with the current simulation approach.

To sum up, almost all the proposed project objectives have been successfully completed, as proves the TCS-3C and this document. The designed and implemented system respects all the proposed requirements and is fully functional. As far as the research side worked, the proposed technique to research of the behaviour of Traffic Participants has been unsuccessful, but further adjustments will be done in order to investigate and find a solution.

8.1. Further Work

First improvement that would be brought to the system is removing the file-based communication between the Live Traffic Server and the Route Computing Server. The map status should be communicated in the body of the request. This step will allow Distributing the system, making it more powerful by using the properties of multiple machines.

After the previous step is done and other minor changes are done, the system would be integrated with a solution like the one presented in [30]. This would prove the real functionality of the system in aiding research of the Connected Cars Context.

Lastly, the current research topic will be further researched in order to find a viable solution that would give the desired results.

Bibliography

[1] Downs, A. (2019). *Traffic: Why It’s Getting Worse, What Government Can Do*. [online] Brookings. Available at: https://www.brookings.edu/research/traffic-why-its-getting-worse-what-government-can-do/ [Accessed 8 Jul. 2019].

[2] Nationwide.com. (2019). *Traffic Jams & Congestion – Nationwide*. [online] Available at: https://www.nationwide.com/road-congestion-infographic.jsp [Accessed 8 Jul. 2019].

[3] Business Insider. (2017). *Here's how much time and money you waste sitting in traffic a year*. [online] Available at: https://www.businessinsider.com/time-money-spent-traffic-per-year-us-cities-new-york-los-angeles-san-francisco-atlanta-2017-2 [Accessed 8 Jul. 2019].

[4] Schneider, B. (2018). *You Can't Build Your Way Out of Traffic Congestion. Or Can You?*. [online] CityLab. Available at: https://www.citylab.com/transportation/2018/09/citylab-university-induced-demand/569455/ [Accessed 8 Jul. 2019].

[5] Miller, S. (2015). *Traffic Congestion: Why It’s Increasing and How To Reduce It*. [online] LivableStreets Alliance. Available at: https://www.livablestreets.info/traffic_congestion_why_its_increasing_and_how_to_reduce_it [Accessed 8 Jul. 2019].

[6] Osmand.net. (n.d.). *OsmAnd - Offline Mobile Maps and Navigation*. [online] Available at: https://osmand.net/ [Accessed 8 Jul. 2019].

[7] GitHub. (n.d.). *Osmand Repository*. [online] Available at: https://github.com/osmandapp/Osmand [Accessed 8 Jul. 2019].

[8] Itinero.tech. (n.d.). *Itinero*. [online] Available at: https://www.itinero.tech/ [Accessed 8 Jul. 2019].

[9] GitHub. (n.d.). *Itinero Repository*. [online] Available at: https://github.com/itinero [Accessed 8 Jul. 2019].

[10] Wiki.openstreetmap.org. (2019). *OpenStreetMap Wiki*. [online] Available at: https://wiki.openstreetmap.org/wiki/Main_Page [Accessed 8 Jul. 2019].

[12] Keijo Ruohonen (2013) Graph Theory. Available at: http://math.tut.fi/~ruohonen/GT_English.pdf [Accessed 8 Jul. 2019].

[13] Luxen, D. (2014). *Smart Directions Powered by OSRM’s Enhanced Graph Model*. [online] Medium. Available at: https://blog.mapbox.com/smart-directions-powered-by-osrms-enhanced-graph-model-3ae226974b2 [Accessed 8 Jul. 2019].

[14] Volker, L. (2008). *Route Planning in Road Networks with Turn Costs*. [ebook] Available at: http://lekv.de/pub/lv-turns-2008.pdf [Accessed 8 Jul. 2019].

[15] Geisberger, C. (n.d.). *Efficient Routing in Road Networks*. [ebook] Available at: https://algo2.iti.kit.edu/download/turn_ch.pdf [Accessed 8 Jul. 2019].

[16] Luxen, D. (n.d.). *Flexible Route Guidance Through Turn Instruction Graphs*. [ebook] Available at: http://algo2.iti.kit.edu/documents/Luxen%20Pub/sigproc-sp.pdf [Accessed 8 Jul. 2019].

[17] Luxen, D. (n.d.). *Smart Directions Powered by OSRM’s Enhanced Graph Model*. [online] Medium. Available at: https://blog.mapbox.com/smart-directions-powered-by-osrms-enhanced-graph-model-3ae226974b2 [Accessed 8 Jul. 2019].

[18] Vergara, V. (2015). *pgRouting/pgrouting*. [online] GitHub. Available at: https://github.com/pgRouting/pgrouting/wiki/Edge-based-Graph-Analysis [Accessed 8 Jul. 2019].

[19] Sawlani, S. (2017). *Explaining the Performance of Bidirectional Dijkstra and A* on Road Networks*. [ebook] Denver. Available at: https://digitalcommons.du.edu/cgi/viewcontent.cgi?article=2303&context=etd [Accessed 8 Jul. 2019].

[20] Abelshausen, B. (n.d.). *RouterDb | Itinero - Documentation*. [online] Docs.itinero.tech. Available at: https://docs.itinero.tech/docs/itinero/basic-concepts/routerdb.html [Accessed 8 Jul. 2019].

[21] Abelshausen, B. (n.d.). *Router | Itinero - Documentation*. [online] Docs.itinero.tech. Available at: https://docs.itinero.tech/docs/itinero/basic-concepts/router.html [Accessed 8 Jul. 2019].

[22] Abelshausen, B. (n.d.). *DynamicVehicle | ItineroAPI Documentation*. [online] Available at: https://docs.itinero.tech/itinero/Itinero.Profiles.DynamicVehicle.html. [Accessed 8 Jul. 2019].

[23] Abelshausen, B. (n.d.). *Content | Itinero - Documentation*. [online] Docs.itinero.tech. Available at: https://docs.itinero.tech/docs/index.html [Accessed 8 Jul. 2019].

[24] Tudorica, A. (2019). *License Repository*. [online] Available at: https://github.com/andreitudorica/UTCN_LICENTA [Accessed 8 Jul. 2019].

[25] Download.geofabrik.de. (n.d.). *Geofabrik Download Server*. [online] Available at: https://download.geofabrik.de/ [Accessed 8 Jul. 2019].

[26] Wiki.openstreetmap.org. (n.d.). *Osmconvert - OpenStreetMap Wiki*. [online] Available at: https://wiki.openstreetmap.org/wiki/Osmconvert [Accessed 8 Jul. 2019].

[27] geojson.io. (n.d.). *Geo Json*. [online] Available at: http://geojson.io [Accessed 8 Jul. 2019].

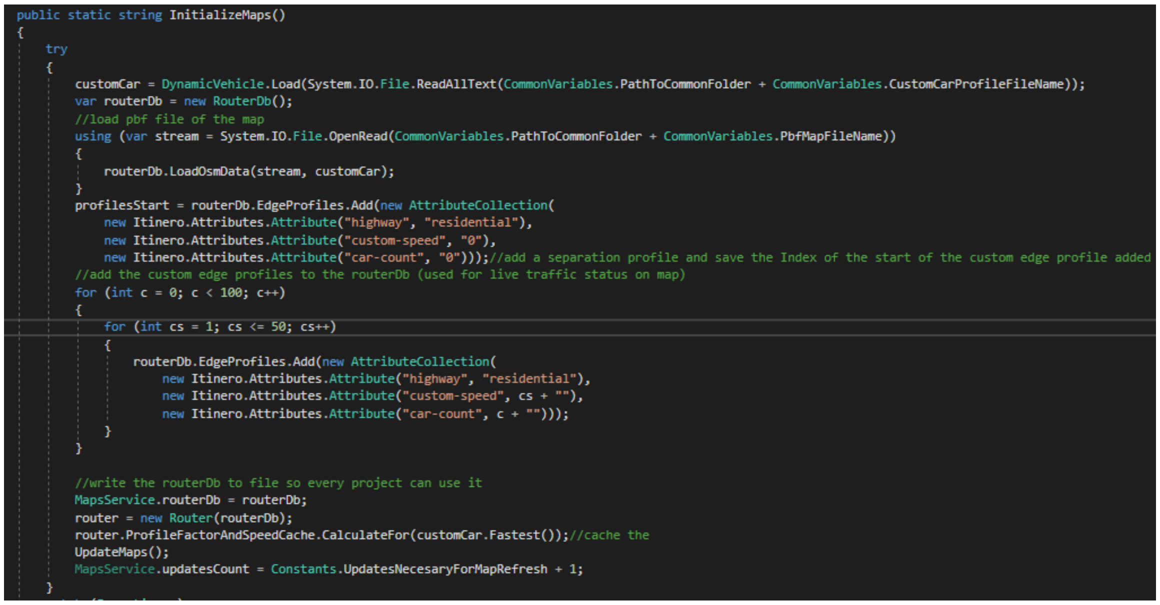
[28] Portforward.com. (n.d.). *Setup a port forward in your router to allow incoming connections for multiplayer games, VOIP, DVR, and security cameras.* [online] Available at: https://portforward.com/ [Accessed 8 Jul. 2019].

[29] Gigi Labs. (n.d.). *Accessing an ASP .NET Core Web Application Remotely*. [online] Available at: http://gigi.nullneuron.net/gigilabs/accessing-an-asp-net-core-web-application-remotely [Accessed 8 Jul. 2019].

[30] Toderici, D. (2018). *Segment Tree Mased Traffic Congestion Avoidance in Connected Cars Context*. Cluj-Napoca.

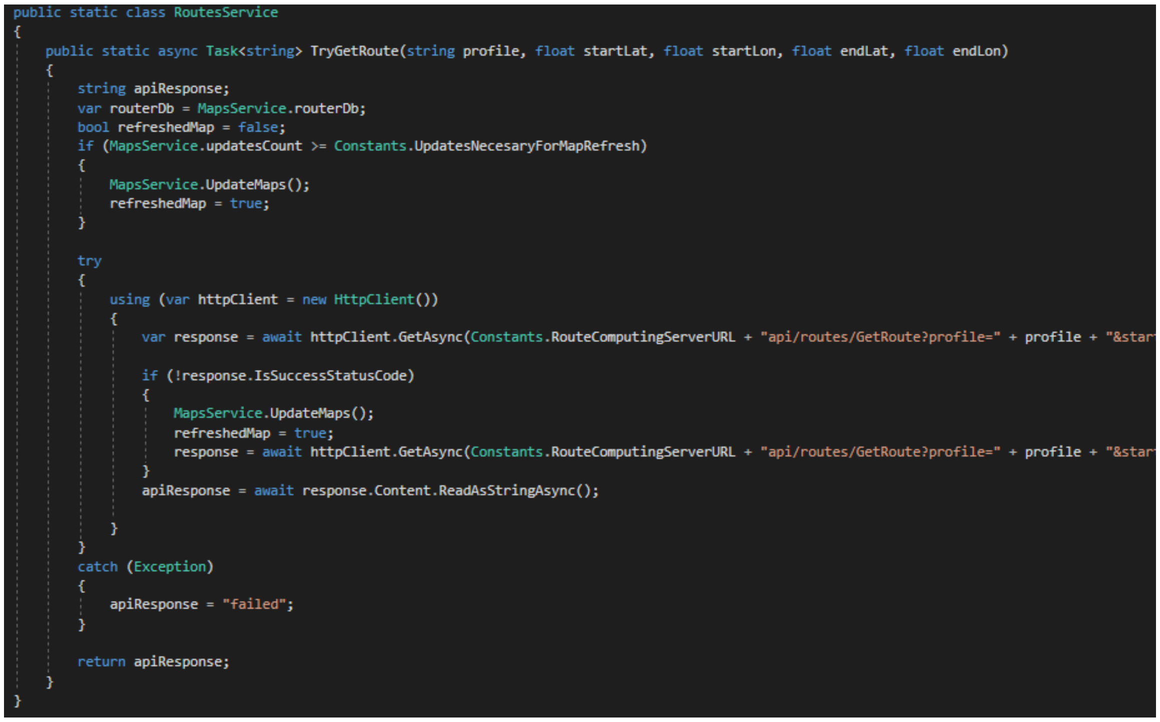
[31] Yoo, J. (2017). *Remote Access to Local ASP.NET Core Applications from Mobile Devices - Kloud Blog*. [online] Blog.kloud.com.au. Available at: https://blog.kloud.com.au/2017/02/27/remote-access-to-local-aspnet-core-apps-from-mobile-devices/. [Accessed 8 Jul. 2019].

Appendix 1 – Relevant code sections



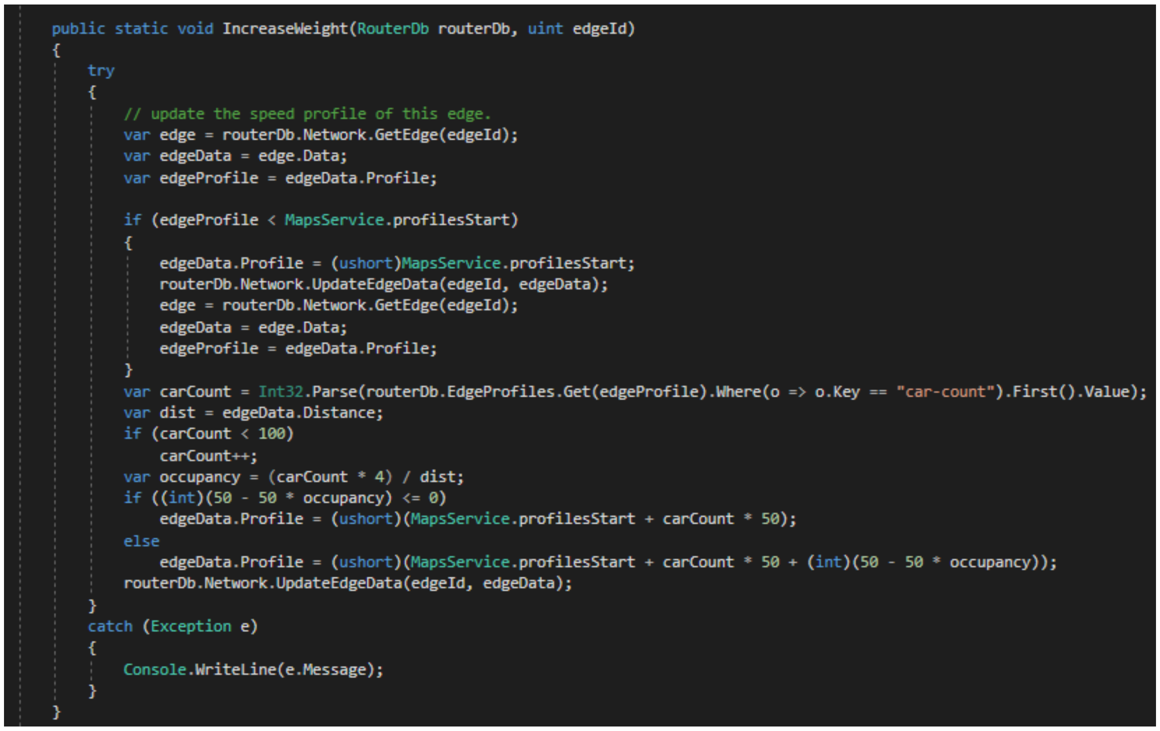
```
public static string InitializeMaps()
{
    try
    {
        customCar = DynamicVehicle.Load(System.IO.File.ReadAllText(CommonVariables.PathToCommonFolder + CommonVariables.CustomCarProfileFileName));
        var routerDb = new RouterDb();
        //load pbf file of the map
        using (var stream = System.IO.File.OpenRead(CommonVariables.PathToCommonFolder + CommonVariables.PbfMapFileName))
        {
            routerDb.LoadOsmData(stream, customCar);
        }
        profilesStart = routerDb.EdgeProfiles.Add(new AttributeCollection(
            new Itinero.Attributes.Attribute("highway", "residential"),
            new Itinero.Attributes.Attribute("custom-speed", "0"),
            new Itinero.Attributes.Attribute("car-count", "0")));//add a separation profile and save the Index of the start of the custom edge profile added
        //add the custom edge profiles to the routerDb (used for live traffic status on map)
        for (int c = 0; c < 100; c++)
        {
            for (int cs = 1; cs <= 50; cs++)
            {
                routerDb.EdgeProfiles.Add(new AttributeCollection(
                    new Itinero.Attributes.Attribute("highway", "residential"),
                    new Itinero.Attributes.Attribute("custom-speed", cs + ""),
                    new Itinero.Attributes.Attribute("car-count", c + "")));
            }
        }

        //write the routerDb to file so every project can use it
        MapsService.routerDb = routerDb;
        router = new Router(routerDb);
        router.ProfileFactorAndSpeedCache.CalculateFor(customCar.Fastest());//cache the
        UpdateMaps();
        MapsService.updatesCount = Constants.UpdatesNecesaryForMapRefresh + 1;
    }
```

Figure A1-1 Screenshot of InitializeMaps method of the MapsService



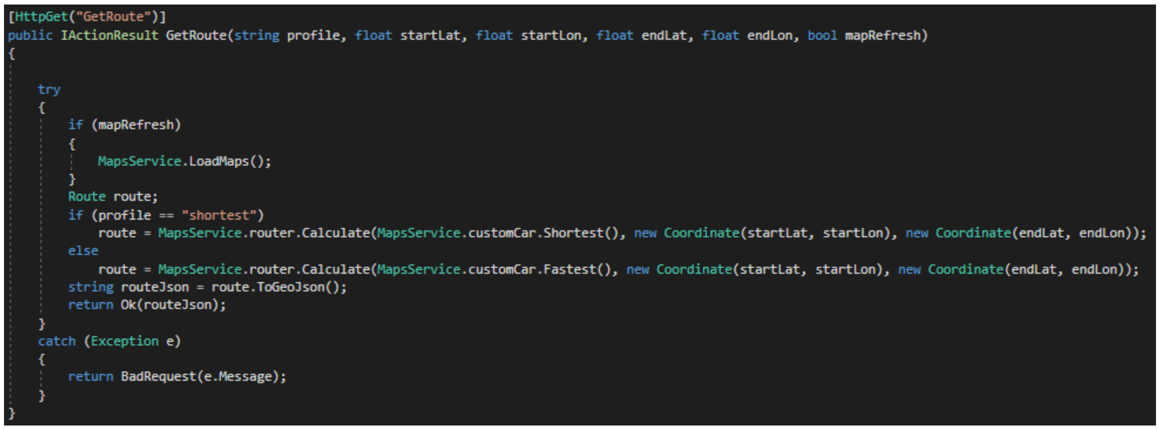
```
public static class RoutesService
{
    public static async Task<string> TryGetRoute(string profile, float startLat, float startLon, float endLat, float endLon)
    {
        string apiResponse;
        var routerDb = MapsService.routerDb;
        bool refreshedMap = false;
        if (MapsService.updatesCount >= Constants.UpdatesNecesaryForMapRefresh)
        {
            MapsService.UpdateMaps();
            refreshedMap = true;
        }

        try
        {
            using (var httpClient = new HttpClient())
            {
                var response = await httpClient.GetAsync(Constants.RouteComputingServerURL + "api/routes/GetRoute?profile=" + profile + "&star

                if (!response.IsSuccessStatusCode)
                {
                    MapsService.UpdateMaps();
                    refreshedMap = true;
                    response = await httpClient.GetAsync(Constants.RouteComputingServerURL + "api/routes/GetRoute?profile=" + profile + "&star
                }
                apiResponse = await response.Content.ReadAsStringAsync();

            }
        }
        catch (Exception)
        {
            apiResponse = "failed";
        }

        return apiResponse;
    }
}
```

Figure A1-2 Screenshot of the TryGetRoute Method in the RoutesService Class



```
public static void IncreaseWeight(RouterDb routerDb, uint edgeId)
{
    try
    {
        // update the speed profile of this edge.
        var edge = routerDb.Network.GetEdge(edgeId);
        var edgeData = edge.Data;
        var edgeProfile = edgeData.Profile;

        if (edgeProfile < MapsService.profilesStart)
        {
            edgeData.Profile = (ushort)MapsService.profilesStart;
            routerDb.Network.UpdateEdgeData(edgeId, edgeData);
            edge = routerDb.Network.GetEdge(edgeId);
            edgeData = edge.Data;
            edgeProfile = edgeData.Profile;
        }
        var carCount = Int32.Parse(routerDb.EdgeProfiles.Get(edgeProfile).Where(o => o.Key == "car-count").First().Value);
        var dist = edgeData.Distance;
        if (carCount < 100)
            carCount++;
        var occupancy = (carCount * 4) / dist;
        if ((int)(50 - 50 * occupancy) <= 0)
            edgeData.Profile = (ushort)(MapsService.profilesStart + carCount * 50);
        else
            edgeData.Profile = (ushort)(MapsService.profilesStart + carCount * 50 + (int)(50 - 50 * occupancy));
        routerDb.Network.UpdateEdgeData(edgeId, edgeData);
    }
    catch (Exception e)
    {
        Console.WriteLine(e.Message);
    }
}
```

Figure A1-3 Screenshot of the IncreaseWeight method of the EdgeWeightsService Static Class



```
[HttpGet("GetRoute")]
public IActionResult GetRoute(string profile, float startLat, float startLon, float endLat, float endLon, bool mapRefresh)
{

    try
    {
        if (mapRefresh)
        {
            MapsService.LoadMaps();
        }
        Route route;
        if (profile == "shortest")
            route = MapsService.router.Calculate(MapsService.customCar.Shortest(), new Coordinate(startLat, startLon), new Coordinate(endLat, endLon));
        else
            route = MapsService.router.Calculate(MapsService.customCar.Fastest(), new Coordinate(startLat, startLon), new Coordinate(endLat, endLon));
        string routeJson = route.ToGeoJson();
        return Ok(routeJson);
    }
    catch (Exception e)
    {
        return BadRequest(e.Message);
    }
}
```

Figure A1-4 Screenshot of the code in the GetRoute endpoint